VŠB – TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY
KATEDRA INFORMATIKY

Porovnání NoSQL databází
Comparison of NoSQL databases

2012 **Michal Ryška**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Michal Ryška**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Porovnání NoSQL databází
Comparison of NoSQL Databases

Zásady pro vypracování:

NoSQL databáze představuje paradigma, které usnadňuje uložení dat v distribuovaném prostředí. Cílem této práce je porovnání dvou existujících NoSQL databází MongoDB a CouchDB využitím benchmarku pro NoSQL databáze. Součástí práce bude také popis hlavních rozdílu mezi těmito databázemi a relačními databázemi.

Práce bude obsahovat následující:
1. Vymezení NoSQL databází vůči relačním databázím.
2. Implementace benchmarku pro NoSQL databáze.
3. Nasazení benchmarku na CouchDB a MongoDB a vyhodnocení experimentů.

Seznam doporučené odborné literatury:

http://www.nosqlbenchmarking.com/

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Radim Bača, Ph.D.**

Datum zadání: 18.11.2011
Datum odevzdání: 04.05.2012

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení:

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 13.7.2012

.....
podpis

Poděkování:
Na tomto místě bych chtěl poděkovat Ing. Radimovi Bačovi, Ph.D za jeho ochotu a vedení mé bakalářské práce.

Abstrakt

Práce se věnuje relativně novým databázovým systémům. Zaměřuje se především na porovnání technologií těchto systémů.
Na začátku je krátce shrnut relační databázový model, jeho základní rysy a nevýhody.
V další části je rozebrána technologie NoSQL, její historie a rozdělení do kategorií.
Následuje část věnovaná databází MongoDB, je zde též nastíněna stručná historie, základní vlastnosti, způsoby dotazování atd.
Další část je věnovaná databázi CouchDB, taktéž je zde uvedena historie, základní principy a nástroje, které databáze využívá.
V navazující části je popsán testovací nástroj Yahoo! Cloud Serving Benchmark, s popisem jeho architektury, a je zde také uveden postup pro samotné spuštění tohoto nástroje.
V části o srovnání databázových systémů jsou zdůrazněny vlastnosti jejich jednotlivých systémů a je zde také uveden výsledek výkonnostního testu.
Závěr je pak věnován shrnutí práce.

Klíčová slova

Relační databázový model, NoSQL, MongoDB, CouchDB, Yahoo! Cloud Serving Benchmark, Dokumentově orientované databáze

Abstract

This study is focuses on a relatively new database systéme. It focuses mainly on comparing the technology of these systems.
At the beginnig, there is a briefly summarized the relational database model, its basic features and disabilities
The next section analyzes the NoSQL technology, its history and categorization,
The following section is devoted to the MongoDB database, there are also outlined a brief history, basic properties, methods of querying, etc.
Another Section is devoted to the CouchDB database, there is also the history, basic principles and tools that use a database.
In the following section there is a description of a testing tool Yahoo! Cloud Serving Benchmark, with a desription of its architecture, and also specifies the procedure for execution of this tool.
In the comparison of database systems are emphasized properties of their systems and there is also a performance test results.
The conclusion is then devotes to a summary of the work.

Keywords

The relational database model, NoSQL, MongoDB, CouchDB, Yahoo! Cloud Serving Benchmark, Dokument-oriented database

Seznam použitých symbolů a zkratek

NoSQL	- Not only Structured Query Language
SQL	- Strucuted Query Language
RDBMS	- Relational Database Management Sytem
ACID	- Atomicity Conzistency Isolation Durability
JSON	- Java Script Object Notaion
HTTP	- Hypertext Transfer Protocol
URL	- Uniform Ressource Locator
REST	- Representational State Transfer
API	- Application Programming Interface
WWW	- Word Wide Web
YCSB	- Yahoo! Cloud Serving Benchmark

Obsah

1.Úvod

V posledních letech se v oblasti databázových systémů stále častěji objevuje pojem NoSQL, který se snaží řešit problémy, které vznikají při pokusech škálovat relační databázové systémy.
Cílem této práce je vysvětlit pojem NoSQL a dále popsat databázové systémy CouchDB a MongoDB a jejich vzájemné porovnání i prostřednictvím testovacího nástroje Yahoo! Cloud Serving Bechmark.Na úvod bude stručně představen relační databázový model a relační databáze.
Pak bude představena technologie NoSQL, její vznik, trocha historie a její kategorie.
Další část se bude věnovat samotnému databázovému systému MongoDB její historie, základní vlastnosti a bude vysvětlen princip pro pozdější porovnání.
Dále bude představen databázový systém CouchDB I u tohoto databázového systému si povíme něco o jejím vzniku a historii a základních vlastnostech.
Následovat bude pár slov o testovacím nástroji Yahoo! Cloud Serving Benchmark pomocí kterého jsou obě databáze testovány.
Na konec oba tyto databázové systémy budou srovnány především podle jejich specifik a oblastí pro které jsou určeny.

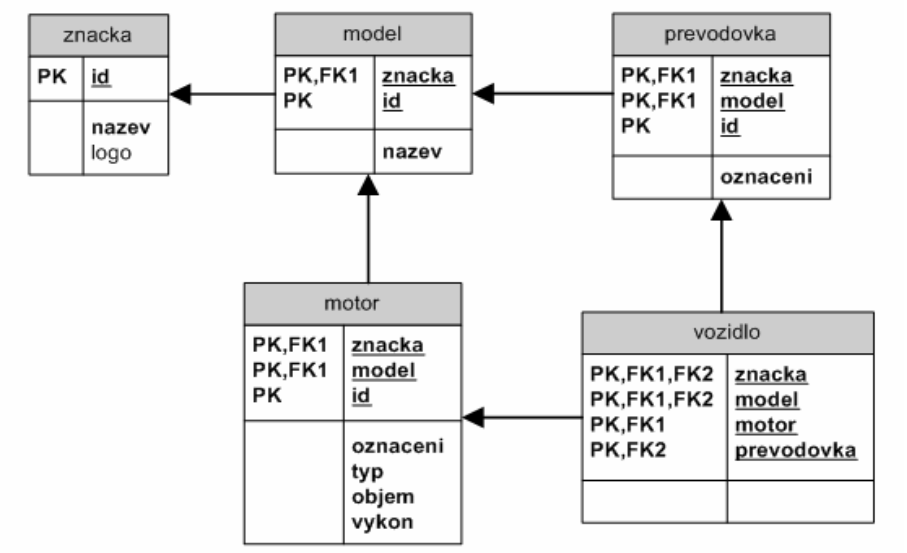
2. Relační databázový model a relační databáze

Relační databázový model sdružuje data do tabulek, které obsahují řádky. Tabulka je jedním ze základních databázových objektů, který slouží k přímému uložení dat do paměťového prostoru relační databáze. Databázovou tabulku si lze představit jako běžnou dvourozměrnou tabulku, která má pevně daný počet a význam jednotlivých sloupců. Každý sloupec má definován jednoznačný název, typ a rozsah. Záznam se stává řádkem tabulky. Pokud jsou v různých tabulkách sloupce stejného typu, pak tyto sloupce mohou vytvářet vazby mezi jednotlivými tabulkami. Kolekce více tabulek a vazeb mezi nimi tvoří relační databázi.

Relační model klade asi největší důraz na zachování integrity dat. Pro dotazování nad relační databází se používá jazyk SQL (Structured Query Language), neboli strukturovaný dotazovací jazyk. Jeho základní model je obecně použitelný pro většinu relačních databází.[1]

2.1 Základní rysy relačních databází

- Databáze obsahuje tabulky. Tabulky obsahují sloupce a řádky. Řádky jsou tvořeny z hodnot sloupců. Všechny řádky v tabulce mají stejné schéma.
- Datový model je vždy předem definován.Databázové schéma je pevně dané a obsahuje vazby a vztahy, které slouží k vynucení integrity dat.
- Datový model je založen na přirozené reprezentaci dat v něm uložených.
- Datový model je normalizován pro odstranění duplicity dat. Normalizace stanovuje vztahy mezi tabulkami a vztahy spojují data mezi tabulkami.



Obrázek č.1: Příklad typického relačního datového modelu

2.2 Problém s relačními databázemi

I když Relační databáze poskytují databázovým uživatelům dobrou kombinaci jednoduchosti, flexibility, výkonu, škálovatelnosti. Kvůli rostoucímu počtu aplikací se jeden z těchto požadavků stává kritickým a pro stále větší počet databázových uživatelů tak důležitým, že zastiňuje všechny ostatní. Tento požadavek je škálovatelnost. V současné době je stále více aplikací spouštěno v prostředí s obrovským zatížením, jako jsou např. webové služby. Jejich požadavky na škálovatelnost se mohou dynamicky měnit a taky velmi rychle růst.
Relační databáze mají dobrou škálovatelnost, ale obvykle pouze v případě, jediného serverového uzlu . Jakmile je dosaženo kapacity tohoto jediného uzlu, je potřeba škálovat a distribuovat zátěž mezi více serverových uzlů, toto řešení bývá v relačních databázích implementačně velmi náročné.
Takže s cílem poskytnout zákazníkům datové úložiště s dobrou škálovatelností vedlo dodavatele databázových systémů k zavedení nového typu databázového systému, který se přímo zaměřuje na škálovatelnost jakožto nejdůležitější vlastnost, na úkor ostatních požadavků relační databáze.

3. NoSQL

Carlo Strozzi poprvé použil termín NoSQL v roce 1998 aby pojmenoval jeho open-source relační databázi, která nebyla vystavěna na standardním SQL rozhraní. Překlad termínu NoSQL zní "ne jen SQL".

Eric Evans tuto zkratku znovu použil v roce 2009, kdy společně s Johanem Oskarssonem zorganizovali konferenci k prodiskutování open-source distribuovaných databází. Na této konferenci poprvé proběhly prezentace databází Voldemort, Cassandra, Dynamite, HBase, Hypertable, CouchDB a MongoDB. Tímto termínem se pokusili označit vznik velkého počtu rostoucích nerelačních, distribuovaných datových úložišť, které se často ani nepokouší poskytnout záruku ACID (Atomicita, Konzistence, Izolace, Trvanlivost), jež jsou klíčové atributy klasických relačních databázových systémů, jako jsou Sybase, IBM DB2, MySQL, Microsoft SQL Server, PostgreSQL, Oracle RDBMS, Informix, Oracle RDB, atd.

3.1 NoSQL

Obecně je NoSQL široká třída databázových systémů, které se liší od klasického modelu relačního databázového systému (RDBMS), v některých významných ohledech.
Termín NoSQL v podstatě znamená, že relační databáze nejsou jedinou možností trvalého řešení, že existují i alternativní možnosti, které jsou v určitých případech vhodnější.
NoSQL databáze se zrodily jako potřeba řešit reálné problémy. Vznik NoSQL databází se vztahuje k projektům, které se musí vypořádávat s obrovským množstvím dat např. Facebook, Google,atd.
V NoSQL databázi není žádné fixní schéma a žádné operace spojení. RDBMS využívá "scales up"(škálování navrch), která spočívá v neustálé obměně hardware a přidávání více pamětí pro rychlejší zpracování dat.
NoSQL využívá oproti RDBMS "scaling out" (škálování ven). Tato výhoda spočívá v rozdělení zátěže mezi více komoditních systémů a tím umožňuje levné řešení pro velké datové soubory.[2]

3.2 Kategorie NoSQL

V současné době je NoSQL rozděleno do čtyř základních kategorií: [2]

3.2.1 Column Family Stores

Column Family Stores jsou velmi podobné tabulce, která funguje jako úložiště pro sloupce a řádky. Nicméně Column Family vyžaduje zásadní posun v uvažování pro ty kteří jsou zvyklý pracovat s relačními databázemi.
V relační databázi definujeme tabulky, které mají definované sloupce. Tabulka určuje jména sloupců a jejich datové typy a každý řádek je tvořen z hodnot sloupců. Všechny řádky v tabulce mají stejné schéma.

Naproti tomu v Column Family Stores definujeme rodiny sloupců (Column families). Column families mohou (a měly by) definovat metadata o jednotlivých sloupcích, ale aktuální sloupce, které tvoří řádek jsou určeny klientskou aplikací. Každý řádek může mít rozdílnou sadu sloupců a každý může mít své vlastní schéma. Každý řádek Column family je jednoznačně definován jeho řádkovým klíčem (row key), je to ekvivalent k primárnímu klíči v relačních tabulkách. Typický příklad Column Family Store je Cassandra, HBase, Hypertable.

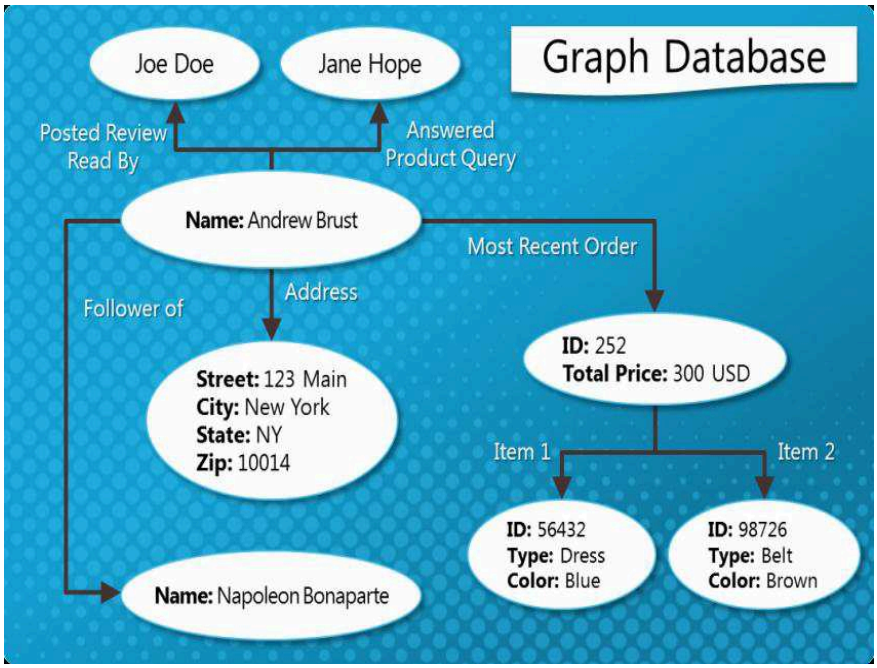
Existují dva typy Column Families:

- Static column family
- Dynamic column family

Obrázek č.2 Příklad Column family stores [blog.msdn.com]

3.2.2 Grafové databáze

Grafové databáze používají teorii grafů k uchování informací o vztazích mezi jednotlivými položkami, jsou obvykle bez schémové a umožňují sadě uzlů (instance objektu) s dynamickými vlastnostmi být libovolně spojeny s dalšími uzly pomocí hran. Databáze má explicitně definované uzly i hrany. Hrany jsou otipované, směrové a jak hrany tak uzly mohou mít definované atributy. Grafové databáze poskytují možnost modelovat velmi komplexní vztahy hojně používané v např. sociálních sítích. Typický příklad grafového databázového systému: Neo4j, AllegroGraph,atd.



Obrázek č.3 Příklad grafové databáze [blog.msdn.com]

3.2.3 Key/Value Store

Distribuované key/value úložiště jsou dnes standardní součástí vysoce výkonných webových služeb a aplikací cloud computing(poskytování služeb či programů uložených na serverech na internetu s tím, že uživatelé k nim mohou přistupovat například pomocí webového prohlížeče nebo klienta dané aplikace a používat je prakticky odkudkoliv).
Key/value úložiště umožňují vývojářům aplikací ukládat data nezávisle na databázovém schématu. Tyto data jsou obvykle složeny z řetězců, které jsou reprezentovány klíčem (key) a aktuálních dat reprezentovaných hodnotou (value). Klíč slouží jako unikátní identifikátor a hodnota buď jako reprezentace aktuálních dat nebo jako ukazatel umístění těchto dat na disku.

Základní rysy Key/Value databáze

- Domény jsou myšleny jako tabulky, ale na rozdíl od tabulek nemusíme definovat žádné databázové schéma .Doménu si můžeme představit v podstatě jako takový pytel, do kterého vkládáme položky. Položky v rámci jedné domény mohou mít rozdílné schémata.
- Položky jsou identifikovány pomocí klíčů, a daná položka může mít k sobě připojenou dynamickou sadu atributů.
- V některých implementacích,mohou být atributy všech řetězcových typů. V jiných mají atributy jednoduchý typ který reflektuje typ kódu jako např. integer, řetězcové pole a seznamy.
- Žádné vztahy nejsou explicitně definovány mezi doménami nebo v dané doméně.
- Podporují jen základní typy operací: find, insert, delete, update

- Prohledávání probíhá za pomocí iterací

Obrázek č. 4 Příklad key/value databáze [blog.msdn.com]

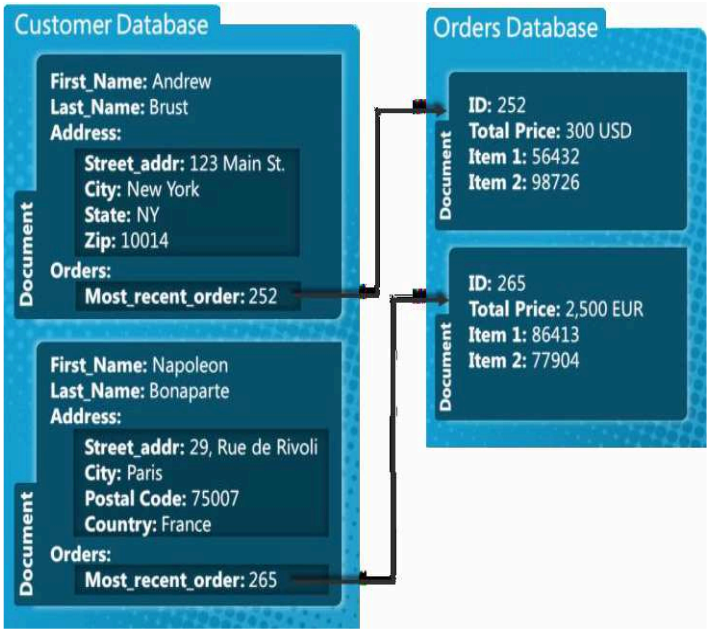
3.2.4 Dokumentově orientované databáze

Dokumentově orientované databáze jsou určeny pro ukládání, získávání, a správu dokumentově orientovaných nebo polo-strukturovaných dat.Dokumentově orientované databáze jsou jedním z hlavních kategorií NoSQL databází. Dokumentové databáze jsou v mnoha ohledech velmi podobné key-value stores, ale liší se v několika podstaných ohledech. Dokumentové databáze podporují více komplexní data než key/value databáze. Dokumentové databáze také pracují s dvojicí klíč – hodnota ale oproti key/values store (kde hodnota může mít libovolný formát) je hodnota považována za dokument kterému databáze rozumí např. JSON, BSON, atd.
Dalším významný rozdíl oproti key-value stores, je že dokumentově orientované databáze obecně podporují sekundární indexy a více typů dokumentů (objektů) na databázi a dále i vnořené dokumenty či seznamy. Stejně jako u ostatních NoSQL systémů, taky dokumentové databáze neposkytují transakční vlastnosti ACID.

Základní rysy Dokumentových databází

- Objekty mohou být uloženy jako dokumenty, stačí pouze převést objektový model na dokument.

- Dokumenty mohou být komplexní, tozn. celý objektový model můžeme přečíst i do něj zapisovat najednou. Není třeba provádět řadu příkazů INSERT nebo vytvářet složité procedury k ukládání.
- Dokumenty jsou na sobě nezávislé, tudíž se zvyšuje výkon a snižuje se možnost souběžností.
- Jsou bez schémové tím pádem poskytují flexibilitu pro vyvíjený systém bez nutnosti přepracování existujících dat.

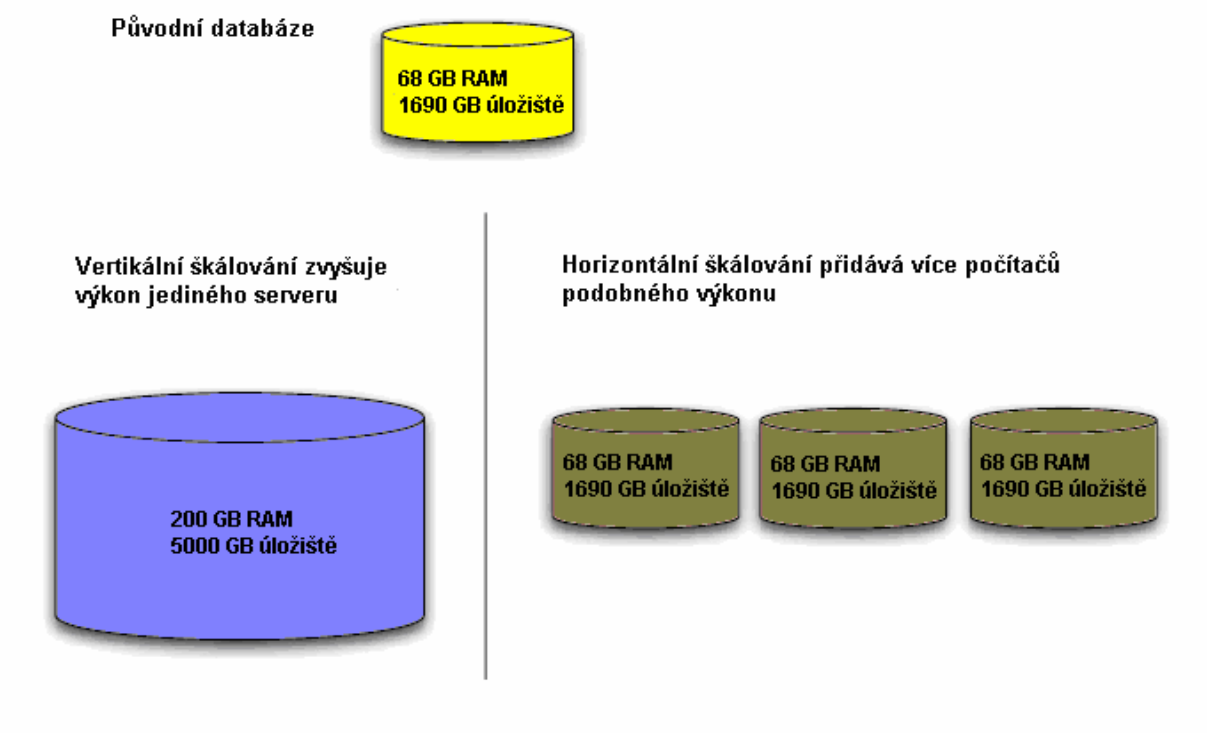


Obrázek č. 5 Příklad dokumentové databáze [blog.msdn.com]

3.3 Škálovatelnost

Nejjednodušší cesta jak škálovat většinu databází je upgrade hardwaru. Pokud aplikace běží na jediném uzlu, je obvykle možné přidat kombinaci přidání pamětí, CPU, nebo disku s větším IOPS. Technika rozšiřování hardwaru jediného uzlu pro škálování je známá jako vertikální škálování. Vertikální škálování má výhodu v tom, že je jednoduché, spolehlivé a úměrné ceně ovšem do určitého bodu. Pokud používáme virtuální hardware (jako např. Amazon EC 2), tak můžeme zjistit, že dostatečně velké instance nejsou k dispozici. Pokud používáme fyzický hardware můžeme se snadno dostat k bodu kdy se náklady na výkonnější server stanou neúnosné.
Pak tedy má smysl uvažovat o horizontálním škálováním, to znamená rozdělení databáze mezi více strojů. Protože architektura horizontálního škálování využívá komoditní hardware, náklady pro hostování celého data setu mohou být významně sníženy, a navíc distribuce dat mezi více počítači zmírňuje následky případného selhání. Pokud požíváme vertikální škálování a dojde k selhání

počítače budeme se muset vypořádat se selháním na němž závisí většina systému. Naproti tomu pokud dojde k selhání v horizontálně škálovatelném modelu, nepředstavuje to až takový problém, neboť jeden počítač představuje mnohem menší procento v celém distribuovaném sytému.



Obrázek č.6 Příklad škálovatelnosti

4. MongoDB

MongoDB je databázový systém, navržený pro webové aplikace a internetové infrastruktury. Datový model a perzistentí strategie je postavena tak, aby umožňovala vysokou propustnost operací čtení a zápis a zároveň schopnost snadně škálovat. MongoDB poskytuje překvapivě dobrý výkon, i když aplikace vyžaduje více databázových uzlů.
Ukazuje se, že databáze MongoDB se stává více atraktivní díky své strategii škálovatelnosti a intuitivnímu datovému modelu. Jestliže dokumentově založený datový model může representovat bohaté, hierarchické datové struktury, je tedy často možné se obejít bez komplikovaných multi-tabulkových spojení (mutli-table joins) vynucených v relačních databázích. S plně normalizovaným relačním datovým modelem může být informace o jednom produktu rozdělena mezi desítky tabulek. A pokud chceme získat reprezentaci daného produktu z databáze, musíme psát komplikované SQL dotazy plné operací spojení. V důsledku toho většina vývojářů budu muset spoléhat na sekundární kus softwaru, který zkompletuje data do něčeho smysluplného.
S dokumentovým modelem, naopak, je většina informací o daném produktu reprezentována v rámci jediného dokumentu. Při použití nástroje MongoDB JavaScript shell můžeme snadno dostat srozumitelnou reprezentaci produktu se všemi jeho informacemi hierarchicky organizovány ve formátu BSON (Binary Java Script Object Notation), a dále s nimi manipulovat či provádět dotazy.Dotazovací funkce jsou speciálně navrženy pro manipulaci se strukturovanými dokumenty.
Většina vývojářů nyní pracuje s objektově orientovanými jazyky a vyžadují datové úložiště s lepším objektovým mapováním. [3]

4.1 Historie MongoDB

Historie dokumentové databáze MongoDB je stručná, ale určitě stojí zato ji aspoň nastínit, protože se zrodila z mnohem ambicióznějšího projektu. V polovině roku 2007 vznikla organizace 10gen a začala pracovat na softwaru "platform-as-a-service", skládajícího se z aplikačního serveru a databáze, která by mohla hostovat webové aplikace a škálovat je podle potřeby. Stejně jako Google´s AppEngine, platforma organizace 10gen byla navržena tak, aby škálovatelnost a správa hardwarové a softwarové infrastruktury probíhala automaticky, díky čemuž se vývojáři mohou zaměřit pouze na vývoj aplikačního kódu. 10gen nakonec zjistila, že se většině vývojářů nelíbí vzdát se kontroly nad svými technologiemi, navzdory tomu ale uživatelé projevili zájem o nové databázové technologie. To vedlo organizaci 10gen k soustředění svého úsilí pouze na vývoj databáze, z které se stala MongoDB.

V současné době 10gen nadále sponzoruje vývoj této databáze jako open source projekt.
Kód je veřejně dostupný a volně upravitelný v souladu s licenčními podmínkami. Velká uživatelská komunita podporuje zasílaní chybových protokolů a opravných patchů.
Přesto, všichni vývojáři jádra MongoDB jsou buď zakladatelé nebo zaměstnanci firmy 10gen,
ale jejich projekt je nadále určován potřebami uživatelské komunity s primárním cílem vytvoření databáze takové, která kombinuje ty nejlepší rysy relačních databází s distribuovanými key/value úložišti.

Tato historie obsahuje pár důležitých myšlenek. První myšlenkou je, že databáze MongoDB byla původně vyvinuta pro platformu, která podle definice, vyžaduje aby databáze byla elegantně škálovatelná na více strojích.

Druhou podstatnou myšlenkou je, že MongoDB byla navržena jako datové úložiště pro webové aplikace. Model databáze MongoDB je horizontálně škálovatelné primární datové úložiště, které se v několika významných ohledech liší od ostatních moderních databázových systémů. [4]

4.2 Základní vlastnosti databáze MongoDB

Databáze je z větší části definována jejím datovým modelem. V této části se podíváme na dokumentový datový model, a také na vlastnosti MongoDB, které nám umožňují efektivně pracovat s tím to modelem.

4.2.1 Dokumentový datový model

Jak už bylo řečeno datový model databáze MongoDB je dokumentově orientován. Co to vlastně dokumenty jsou v souvislosti s databází lze nejlépe demonstrovat příkladem.

```
{ _id: ObjectID('4bd9e8e17cefd644108961bb'),
  title: 'Adventures in Databases',
  url: 'http://example.com/databases.txt',

  author: 'msmith',
  vote_count: 20,

  tags: ['databases', 'mongodb', 'indexing'],

  image: {
    url: 'http://example.com/db.jpg',
    caption: '',
    type: 'jpg',
    size: 75381,
    data: "Binary"
  },

  comments: [
    { user: 'bjones',
      text: 'Interesting article!'
    },

    { user: 'blogger',
      text: 'Another related article is at http://example.com/db/db.txt'
    }
  ]
}
```

_id pole je primární klíč

1 Tagy jsou uloženy jako pole řetězců

2 Atributy dalšího dokumentu

3 Komentáře uložené jako pole objektů

Obrázek č.7 Dokumentový datový model

Obrázek ukazuje vzorový dokument, který představuje článek na socialním webu. Jak můžete vidět, dokument je v podstatě soubor názvů vlastností a jejich hodnot. Hodnoty mohou být jednoduchého datového typu, jako řetězce, čísla a data. Ale hodnotami také mohou být pole a do konce i jiné dokumenty. Obě tyto konstrukce umožňují dokumentům representovat řadu bohatých datových struktur. Dále vidíme že dokument má vlastnosti (property tags), které ukládají tagy jako pole. Ale ještě zajímavější jsou vlastnosti komentářů, který odkazuje na pole s komentovanými dokumenty. Co je důležité si uvědomit, je, že dokumentově orientovaný datový model přirozeně representuje

data v agregované formě, což umožňuje pracovat s objektem jako s celkem. Všechny data od komentářů po tagy mohou být vloženy do jediného databázového objektu.

Databáze MongoDB seskupuje dokumenty do kolekcí, které nevyžadují žádný druh schématu. Teoreticky může každý dokument v kolekci mít kompletně odlišnou strukturu a to nám poskytuje určité výhody. Za prvé aplikační kód, a ne databáze, využívají datové struktury. To může urychlit počáteční vývoj aplikací kdy se schéma poměrně často mění. Za druhé, což je významnější, bez schémový model umožňuje representovat data s variabilními vlastnostmi.

4.2.2 Ad hoc dotazování

Pokud chceme říct, že systém podporuje ad hoc dotazování, říkáme tím není nutné předem definovat jaké druhy dotazů bude systém akceptovat.
Jedním z hlavních cílů návrhu databáze MongoDB je zachování většiny dotazů, které jsou tak neodmyslitelně spjaty s relačním databázovým světem. Jak vlastně dotazovací jazyk databáze MongoDB funguje je ukázáno v následujícím příkladu. Předpokládejme že chceme najít všechny příspěvky s tagem politics s více než deseti hlasy.

SQL dotaz bude vypadat takto:

```
SELECT * FROM posts
  INNER JOIN posts_tags ON posts.id = posts_tags.post_id
  INNER JOIN tags ON posts_tags.tag_id == tags.id
  WHERE tags.text = 'politics' AND posts.vote_count > 10;
```

Obrázek č.8 SQL Dotaz

Ekvivalentní dotaz v MongoDB bude vypadat takto (tag $gt znamená větší než podmínka):

```
db.posts.find({'tags': 'politics', 'vote_count': {'$gt': 10}});
```

Obrázek č.9 Dotaz v mongoDB

Každý z těchto dotazů přejímá jiný datový model. SQL dotaz se striktně opírá o normalizační model, kde jsou příspěvky a tagy uloženy v různých tabulkách, zatímco dotaz v MongoDB předpokládá že tagy jsou uloženy v uvnitř příspěvku daného dokumentu. Ale oba dotazy prokazují schopnost dotázat se na libovolnou kombinaci vlastností, což je podstatou ad hoc dotazování.
Ale samotné ad hoc dotazování nestačí. Jakmile začne data set růst do určité velikosti, indexy se stávají nezbytnými pro efektivní a rychlé dotazování. Indexy řádově zrychlují dotazování a třídící schopnosti. V důsledku toho by každý systém, který podporuje ad hoc dotazování měl také podporovat sekundární indexy.

4.2.3 Sekundární indexy

Databázový index je jakákoliv datová struktura, která uspořádá hodnoty jednoho či více zvolených atributu za účelem rychlejšího prohledání, případně řazení, záznamu na těchto atributech. Nevýhodou je nutnost udržovat tuto strukturu aktuální při každém vložení, upravení nebo odmazání

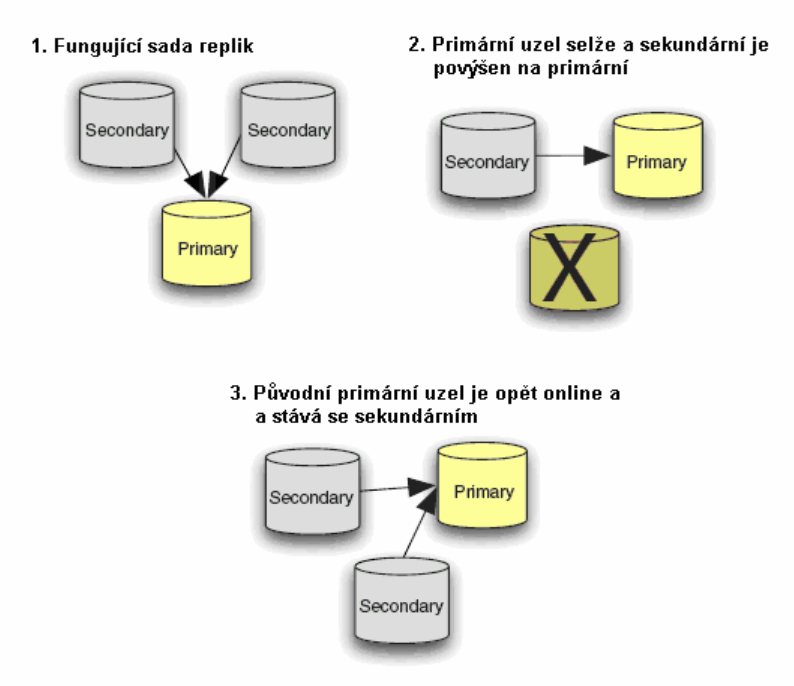
záznamu, což se sebou přináší zvýšený počet zápisu na vnější paměť. K zjednodušení vyhledávání se používají sekundární indexy, které dovolí nesouvisející záznamy přeskakovat.
Sekundární indexy v databázi MongoDB jsou implementovány jako B-stromy. B-stromové indexy jsou implicitní pro většinu relačních databází, jsou optimalizovány pro množství dotazů, včetně skenování (range scan) a třídění. Povolením mnohonásobných sekundárních indexů umožňuje MongoDB uživatelům se přizpůsobit pro širokou řadu dotazů.
V databázi můžete vytvořit až 64 indexů na jednu kolekci. Podporované druhy indexů jsou všechny které můžeme najit i v relačním databázovém vedoucím sytému (RDBMS) tedy: rostoucí, sestupný, unikátní, složený-klíč. Protože MongoDB a většina RDBMS používají stejné datové struktury, správa indexů v obou těchto systémech je kompatibilní.

4.2.4 Replikace

Replikace je proces kopírování a údržby databázových objektů ve více databázích, které tvoří distribuovaný databázový systém. Změny aplikované v jedné databázi jsou zachyceny a uloženy lokálně před posláním a aplikováním na každou databázi v distribuovaném databázovém sytému. Replikace poskytuje uživatelům snadný a rychlý přístup k replikovaným datům.
MongoDB provádí databázovou replikaci přes topologii zvanou sada replik (replica set). Sada replik distribuuje data mezi stroje kvůli zálohování a automatické obnově pro případ výpadku serveru nebo celé sítě.
Sady replik sestávají z jednoho primárního uzlu a z jednoho či více sekundárních uzlů. Stejně jako master-slave replikace, které používají jiné databázové systémy, může primární uzel sloužit jak ke čtení tak i k zapisování, ale sekundární uzly mohou sloužit pouze a jen ke čtení (read only). Co ale činí sadu replik tak unikátní je podpora automatické obnovy při výpadku. Jestliže dojde k výpadku primárního uzlu cluster automaticky vybere jeden ze sekundárních uzlů a povýší ho na uzel primární, a pokud se bývalý primární uzel vrátí zpět k činnosti stane se z něj uzel sekundární.

4.2.5 Operace nad replikacemi

Jak už bylo výše zmíněno MongoDB používá Master – Slave replikaci, tedy číst mohou všechny uzly ale o veškeré zapisování se stará řídící uzel. Ten poté zapisuje data do databáze. Klient si může ověřit, že jsou data korektně zapsána potvrzujícím požadavkem – getLastError. K zapsání dat do databáze dochází až po ukončení celé transakce (transakce jsou pouze atomické a probíhají v rámci jednoho dokumentu) a až po operaci Commit. Tedy v jednu chvíli se data v různých uzlech mohou lišit a může dojít k načtení neaktuálních dat. Proto je nutné vyčkat až do ukončení celé transakce. Pokud se v danou chvíli nacházejí v databázi neaktuální data a je obdržen požadavek na jejich modifikaci či přepis databáze vygeneruje chybové hlášení o neaktuálnosti dat a provede operaci Rollback. Dále pokud existují data, která se nereplikovaly z primárního uzlu (např. kvůli výpadku serveru) jsou tyto data ztraceny.



Obrázek č.10 Replikace

4.2.6 Shrnutí

Abychom to shrnuli, tak Mongodb (odvozeno od "Humongous") je škálovatelný, vysoce výkonný, open source, dokumentově orientovaný databázový systém napsaný v jazyce C++.
Z pokročilých funkcí MongoDB podporuje indexy, agregační funkce, replikaci či odkazy mezi jednotlivými kolekcemi. Databáze je navržena pro moderní internetové aplikace a je připravena k uchovávání velkých souborů, a také pro běh v prostředí vyžadující vysokou dostupnost.
Ovšem MongoDB má i své nevýhody. Jednou z nevýhod je chybějící podpora komplexních víceřádkových transakcí (jednoduché transakce podporují) a tyto transakce je třeba řešit na úrovni aplikace. Je tomu tak z toho důvodu, že tyto operace je složité poskytnout na distribuovaném systému. Další nevýhodou databáze MongoDB je, že nepodporuje vlastnosti ACID ani operace spojení.

5. CouchDB

CouchDB je dokumentově orientovaný databázový systém, vydaný pod open source licencí firmy Apache. Na rozdíl od většiny databázových systémů jsou data ukládána bez potřeby jakéhokoliv schématu. To znamená, že na rozdíl od tradičních SQL databází zde nejsou žádné tabulky a sloupce, primární nebo cizí klíče, operace spojení ani žádné vztahy.
Místo toho, databáze CouchDB ukládá data do série dokumentů a nabízí Java-Skriptově pohledově založený model (JavaScript-based view) pro agregaci a získávání dat. Jméno CouchDB je zkratka pro "Cluster Of Unreliable Comodity Hardware" a je tím naznačeno, že CouchDB je určena k běhu na levných distribuovaných serverech. Každý kdo se zabýval replikacemi v databázích ví, že to je jen výjimečně jednoduchý úkol, nikoliv však pokud se jedná o CouchDB. Když k tomu přidáme skutečnost je CouchDB je napsána v Erlang OTP, programovacím jazyku tolerantním k chybám a který nabízí vynikající funkce souběžnosti díky nimž se databáze dá snadno škálovat bez ztráty spolehlivosti a dostupnosti.
V současné době je CouchDB k dispozici pro většinu UNIX-ově založených systémů včetně LINUX a Mac OSX. Binární instalátory jsou dostupné pro Ubuntu, Fedora, Pentos, atd. prostřednictvím individuálních ovladačů, ovšem podpora pro systém Windows je dost povrchní.

5.1 Historie CouchDB

V dubnu roku 2005, Damien Katz zveřejnil na svém blogu zprávu o vytvoření nového databázového nástroje. V této ranné fázi nebylo k dispozici moc podrobností, ale na svém blogu se vyjádřil, že to bude úložný systém pro objemné škálovatelné objektové databáze, a že se bude nazývat CouchDB. Jeho cílem bylo vytvoření databáze takové, aby byla od základu navržena pro internet a webové aplikace. Katz tedy začal brzy po uveřejnění jeho příspěvku pracovat na této databázi a zvolil si jazyk C++ jako platformu pro její vývoj.
Už od samého začátku byla CouchDB navržena jako bezschémová a indexovatelná pomocí kombinace dodatkového úložiště (append-only storage) a atomických aktualizací. Bylo jasné že Katz se inspiroval modelem Lotus Notes, produktem na kterém pracoval po mnoho let. Pojem dodatkové úložiště znamená že data v databázi CouchDB nebudou nikdy přepsána, ale spíše se označí jako zastaralé, s tím že novější data dostanou přednost.
V listopadu 2005, Katz oznámil, že pracoval na jazyku Fabric formula. Kazt již byl dříve zapojen do vývoje jazyka Lotus Notes Formula, z kterého zdědil Fabric mnoho funkcí.
V prosinci roku 2005 Katz zveřejnil další příspěvek kde nastínil své cíle a ambice pro CouchDB, říkající že vytvořil nový Lotus Notes postavený od základu pro internet. V tom to příspěvku se dostali do popředí vlastnosti, které jsou k dispozici v CouchDB dnes, jako dokumentová orientace, distribuovaná architektura, obousměrná replikace a offline přístup. Díky tomu Damien Katz doufal, že se z CouchDB stane databázový nástroj pro aplikace jako e-mail, vyhledávání chyb či pro blogy a RSS kanály.
Velkým milníkem ve vývoji CouchDB bylo prohlášení z února 2006, ve kterém oznámil, že vývoj databáze bude pokračovat pouze v jazyce Erlang. Tento programovací jazyk byl vyvinut firmou Ericsson a je masivně používaný v telekomunikačním průmyslu. Hlavní vlastností jazyku Erlang je kontrola souběžnosti, dále pak tolerance chyb a podpora pro distribuované aplikace.

Další průlom přišel v dubnu 2006, kdy bylo oznámeno, že CouchDB by mělo být přístupné výhradně přes na HTTP založeném aplikačním rozhranním. To znamená, že spíše než připojením k databázovému serveru využívající klientskou aplikaci, můžete použít jakýkoli software schopný komunikovat s HTTP webovým serverem k vytvoření požadavku, který by následně prováděl databázové akce vracejíc odpovídající reakci.Takže vlastně můžeme databázi jednoduše spravovat navštívením URL ve webovém prohlížeči, a pomocí nástrojů příkazového řádku jako Curl, a co víc přes jakýkoliv programovací jazyk s podporou HTTP požadavků. První veřejně dostupná verze 0.2 CouchDB byla k dispozici ke stažení v srpnu 2006. [5]

5.2 CouchDB – Dokumentově orientovaná databáze

Klíčovou vlastností CouchDB je, že se jedná o dokumentově orientovaný řídící systém. V podstatě to znamená, že data uložená v databázi CouchDB zahrnují sérii dokumentů z nichž každý obsahuje řadu polí a hodnot. Všechny dokumenty jsou na sobě nezávislé, a neexistuje zde žádné striktní schéma, které by měli dodržovat. Tradiční databáze, které se drží relačního modelu dat ukládají data do série tabulek, které jak už bylo řečeno sestávají z řádků a sloupců.
SQL databáze obsahují vztahy, primární klíče, cizí klíče, referenční integrity, atd. Tyto pojmy v databázi CouchDB neexistují.
Místo primárního klíče každý dokument v CouchDB má unikátní ID. Toto unikátní ID může být přiřazeno buď uživatelem nebo aplikací, nebo může být použit univerzální unikátní identifikátor (UUID)- náhodné číslo generované databází CouchDB, které výrazně snižuje šanci na použití duplicitních ID. Všechny data související s daným dokumentem jsou uloženy právě v tomto dokumentu. Skutečnost, že CouchDB nemá žádné pevně dané schéma je velmi důležitá. Při vývoji relačních databází musíme pečlivě přemýšlet o tom jak databázi budeme modelovat předtím než ji vůbec začneme vytvářet, navíc je třeba řešit řadu závislostí a integritních omezení.V databázi CouchDB je každý dokument nezávislý, takže není třeba ukládat redundantní nulové hodnoty, a můžeme definovat nová pole pro každý dokument nezávisle na sobě či dalších dokumentech.
Pochopitelně, že bez-schémová architektura má také pár nevýhod, jak už z názvu vypovídá je to absence schématu, což může být pro některé vývojáře zvyklé pracovat s relačními databázemi problém a také nadbytečná replikace dat mezi dokumenty. Vývojáři ovšem otevřeně konstatovali, že CouchDB nemá být přímou náhradou za relační SQL databáze. Místo toho vidí CouchDB jako alternativní řešení pro případy kdy by mělo smysl dokumentově orientovanou architekturu použít. V aplikacích jako je Wiki, systémy pro správu dokumentů, diskusní fóra, blogy, atd.

5.3 Dokumenty v CouchDB

Data v databázi CouchDB jsou uložena jako série unikátně pojmenovaných dokumentů. Hodnoty uložené v dokumentu mohou být typu string, numbers, dates, booleans, a dalších. Jak už také bylo zmíněno, každý dokument v databázi má své unikátní ID, a dokumenty jsou uloženy v databázi v jednotném adresovém prostoru. Není nijak omezen počet polí, dokumenty mohou být tak velké, jako jsou hodnoty v nich uložené. K datovým polím je nutno dodat, že každý dokument obsahuje metadata, která jsou spravována samotným CouchDB serverem jako je revize čísel a další.
Použití revize dokumentů je důležitá, protože CouchDB sama o sobě nevkládá na data žádný uzamykající mechanismus. Pokud dva uživatelé upravují ty samá data v ten samý čas, prvnímu z nich se podaří potvrdit (commit) aktualizaci bez problémů, zatímco druhý obdrží chybové hlášení o konfliktu. Pokud je zjištěn konflikt uživateli bude nabídnuta poslední revize dokumentu s nabídkou možnosti provést změny v databázi znovu.Také je důležité poznamenat, že CouchDB

nebude nikdy přepisovat existující dokumenty ale přidá nový dokument do databáze s tím, že poslední revize dokumentu se stane prominentní a ostatní jsou uloženy pro archivní účely.
Struktura CouchDB a potvrzovací systém se drží vlastností ACID.

5.4 Nástroj JavaScript View

Protože architektura databáze CouchDB je bez-schémová data jsou uloženy velmi nestrukturovaně. I když to znamená, že data jsou flexibilní a jednotlivá pole lze snadno měnit bez porušení integrity databáze, také to znamená, že může být obtížnější získat informace o jednotlivých datech.
Pro tyto případy CouchDB poskytuje nástroj JavaScript View založený na modelu Spidermonkey, který umožňuje vytvořit ad hoc pohled, ve kterém je možno provádět agregace a operace spojení, což nám umožní získat informace o jednotlivých datech v dokumentech. Tyto pohledy nejsou fyzicky uložené v databázi a v důsledku toho nemají vliv na aktuální data. Není určený žádný limit pro počet pohledů, které můžeme použít.
V databázi Couchdb jsou pohledy definovány uvnitř návrhového dokumentu, a stejně jako datové dokumenty mohou být replikovány jednotlivými instancemi databáze. V CouchDB je každý pohled vytvořen JavaScript funkcí, která tvoří Map část MapReduce operace. Funkce transformuje dokument v jedinou hodnotu, kterou poté vrací. Lze také vytvářet dočasné pohledy ovšem z výkonnostních důvodů to není doporučeno.
Vždy když klient provede požadavek na čtení, CouchDB se ujistí, že požadovaný pohled je aktuální. Pokud požadovaný pohled neobsahuje nejnovější změny v databázi, jsou tyto úpravy postupně vkládány do tohoto pohledu. Tento systém zmírňuje výkonnostní problémy spojené s dynamickým generováním pohledu po každé, když pohled spustíme, a to zejména v databázích které ukládají miliony záznamů.

5.5 REST HTTP aplikační programovací rozhranní

Tradiční relační databáze jsou typicky přístupné pomocí klientských softwarových aplikací a databázové transakce se uskutečňují pomocí strukturovaného dotazovacího jazyka SQL. CouchDB používá odlišný způsob závisející na REST-ful HTTP aplikačním programovacím rozhranním poskytujícím uživatelům přístup k datům. Pro pozdější pokračování je třeba vysvětlit zkratky.
REST (Representational State Transfer) je architektura pro distribuované prostředí a v tomto smyslu odkazuje na skutečnost, že přístup k datům je k dispozici prostřednictvím řady jednoduchých webových služeb, které jsou implementovány v HTTP a drží se principů REST.
JavaScriptová Objektová Notace (JSON) je odlehčený textově založený datový formát pro reprezentaci dat. Aplikační programovací rozhranní (API) je základní rozhranní nabízené vývojářům pro vytváření aplikací a pro interakci s databází. Typické programovací jazyky mají klientskou knihovnu pro interakci s konkrétní databází. Ačkoli i pro databázi CouchDB existují klientské knihovny (výhradně za účelem snadnější interakce s databází), jakákoli platforma, která podporuje HTTP požadavky může komunikovat s CouchDB bez nutnosti instalování dodatečných knihoven.
Při použití CouchDB API je vyslán HTTP požadavek na server CouchDB. Co server s požadavkem udělá závisí na URI (jednotném identifikátoru zdroje). CouchDB jedná se všemi uloženými částmi jako se zdroji. Každá část má svou unikátní URI dostupnou přes HTTP. REST využívá HTTP metod POST, GET, PUT a DELETE pro čtyři základní CRUD (Create, Read, Update, Delete) operace nad všemi zdroji. Když server dokončí zpracování požadavku vrátí data (nebo podrobnosti o chybě pokud nějaká nastala), nebo vhodnou odpověď ve formátu JSON.

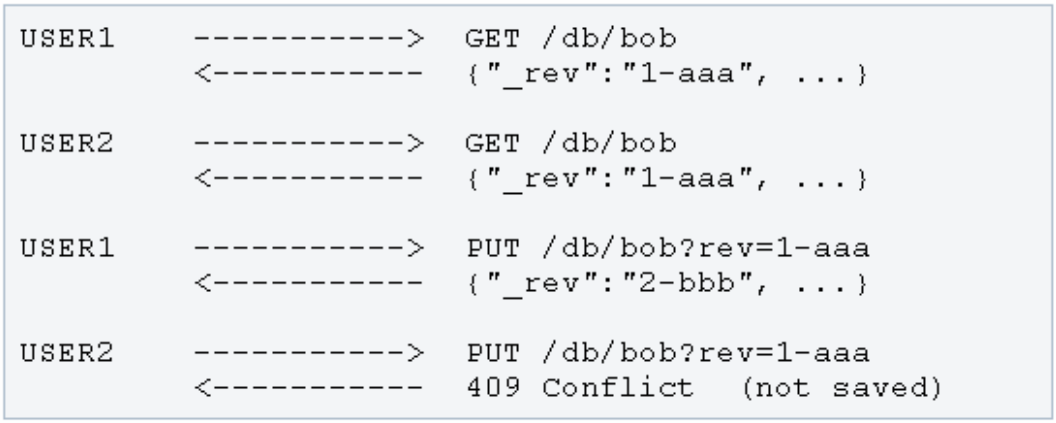
Parsování JSON je poměrně jednoduché, a pomocí JavaScript framework (jako je Prototype či jQuery) vytváří odesílání HTTP požadavků a parsování JSON odpovědí velmi jednoduše prostřednictvím metod Ajax.

5.6 Replikace

CouchDB je navržena pro spolehlivou obousměrnou replikaci – tedy Master – Master, zapisovat i číst data mohou všechny uzly. Dociluje toho prostřednictvím inkrementálního replikovaného modelu, kde jsou zpracovány pouze ty dokumenty, které se změnily od poslední replikace. Systém replikací v databázi CouchDB umožňuje replikovaným procesům, které selhaly se znovu obnovit z posledního uloženého záchranného bodu. Kromě obyčejných CouchDB dokumentů, které ukládají data, návrhového dokumenty které obsahují CouchDB pohledy mohou být také replikovány, stejně jako přílohy jakéhokoli jiného dokumentu. To znamená, že ne jen data ale všechny aplikace v CouchDB mohou využívat funkce replikace.

5.6.1 Operace nad replikacemi

Veškeré operace probíhají v rámci jediného dokumentu a jediného uzlu. CouchDB udržuje konzistenci mezi více databázemi prostřednictvím inkrementální replikovaného modelu, tedy všechny databázové servery zůstávají v nepřetržité komunikaci. Inkrementální replikace je proces, kde jsou se změněné dokumenty periodicky kopírovány mezi servery. Tedy každý uzel je nezávislý a soběstačný a obsahuje aktuální data. Jak už bylo řečeno CouchDB používá obousměrnou replikaci (zapisovat i číst data mohou všechny uzly) a díky inkrementálnímu replikovanému modelu jsou prováděny všechny operace nad aktuálními daty. Řešení konfliktu probíhá následovně.Databáze CouchDB zabraňuje vzniku konfliktu vrácením chyby 409 HTTP error. Když vložíme novou verzi dokumentu musíme vyplnit pole _rev předchozí verze. Pokud už bylo pole _rev vyplněno a nahrazeno, je aktualizace odmítnuta s chybou 409. Níže je uveden příklad pro pochopení.
Příklad: Dva uživatelé na stejném uzlu si stáhnou Bobovu vizitku aktualizují ji a změny zapíšou zpět.



Obrázek č. 11 Přiklad konfliktu

5.7 Futon

Futon je online nástroj pro správu CouchDB, který umožňuje uživatelům interakci s databází.
Tato webová aplikace je zabalena v každém instalačním balíčku a běží na stejném serveru jako CouchDB. V důsledku toho je Futon automaticky k dispozici při spuštění databáze. Tento nástroj nabízí možnost vytvářet, upravovat či mazat databáze a dokumenty a také třídící i stránkovací funkce. Umožňuje také snadno komprimovat databáze a spouštět řadu diagnostik pro správný běh databáze.

Obrázek č.12 Nástroj Futon

6. Yahoo! Cloud Serving Benchmark

Většinou bývá dost obtížné rozhodnout, který systém je vhodný pro vaši aplikaci, částečně proto, že systémy mají odlišné funkce a částečně také proto, že není snadné najít způsob jak porovnat výkon jednoho systému oproti druhému. Cílem projektu YCSB je vyvinout rámec (framework) a společné soubory zatížení (workloads) pro vyhodnocení výkonnosti různých "key-value" a "cloud" databázových systémů.

YCSB projekt sestává ze dvou věcí:

- YCSB klient - rozšiřitelný generátor souborů zatížení
- Jádro souborů zatížení (Core Workloads) - sady zátěžových scénářů, jež mají být vykonány generátorem

Ačkoli definované soubory zatížení poskytují dobrý obraz o výkonnosti systému, samotný klient je rozšiřitelný, takže je možné definovat nové a odlišné soubory zatížení k prozkoumání aspektů systému, nebo aplikačních scénářů, které nejsou součástí jádra souborů zatížení. Stejně tak je klient rozšiřitelný pro testování různých databází.
YCSB nástroj je určen pro testování a porovnávání mnoha systémů. Například můžeme nainstalovat více systémů na stejné hardwarové konfiguraci a spustit ten samý soubor zatížení v každém systému. Poté můžeme vykreslit výkon každého sytému a porovnat v čem jsou jednotlivé systémy lepší. [6]

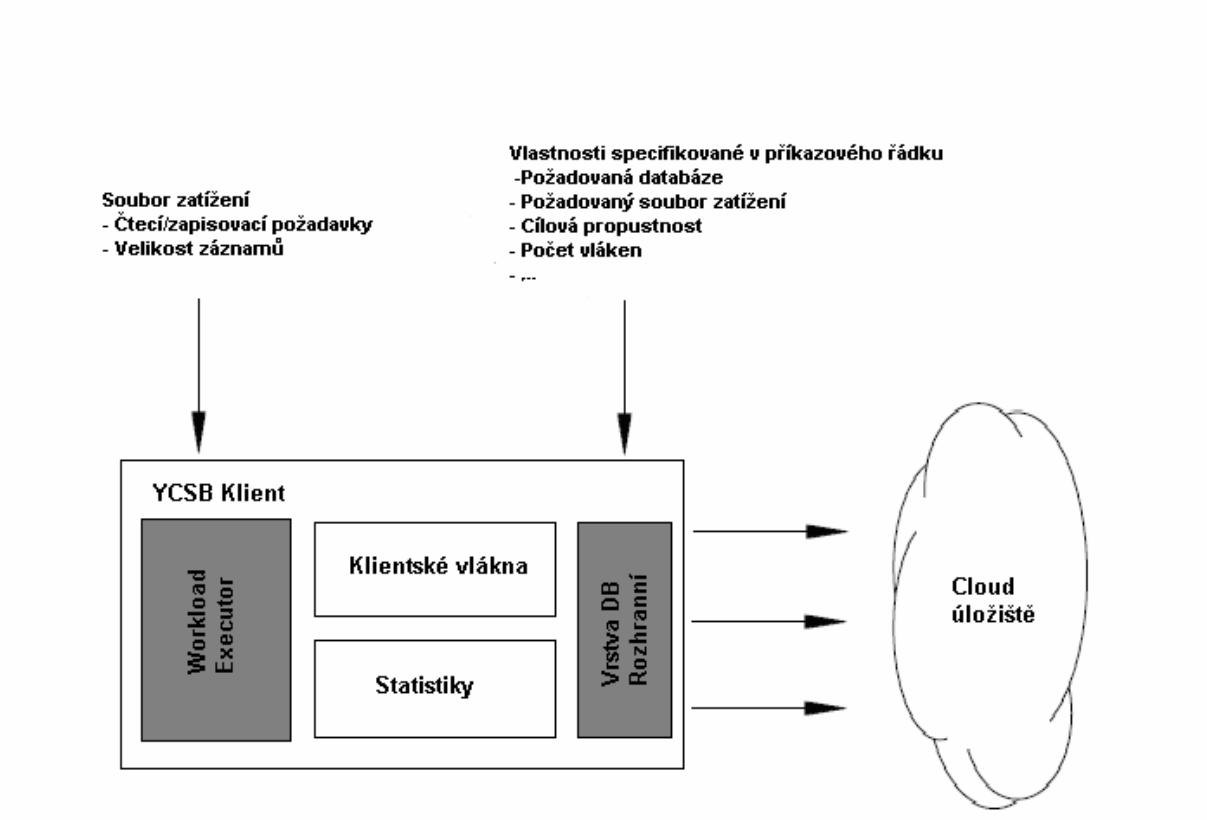
6.1 Architektura

YCSB klient je program napsaný v jazyce Java pro generování dat, které mají být načteny do databáze a pro generování operací, které tvoří jednotlivé soubory vytížení (workloads). Mechanismus zvaný "workload executor" se stará o základní činnost řízení více klientských vláken. Každé vlákno vykonává sérii sekvenčních operací vytvořením požadavků na databázovou rozhranní vrstvu (database interface layer). Prvním je načtení databáze (načítací fáze) a druhým je vykonání samotných souborů zatížení (transakční fáze). Vlákna kontrolují rychlost, s kterou vytvářejí požadavky. Vlákna také měří latenci a dosažitelnou propustnost operací, a vrací tyto naměřené hodnoty do statistického modulu. Na konec statistický modul agreguje měření a vrací propustnost dat, průměrný devadesátý pátý a devadesátý devátý percentil latencí i histogram s jednotlivými prodlevami. Klient přebírá sérii vlastností (páry jmen/hodnot), které definují jeho činnost.

Tyto vlastnosti jsou rozděleny do dvou skupin:

- Vlastnosti zatěžovacích souborů – Tyto vlastnosti definují zatěžovací soubory nezávisle na dané databází. Například mix read/write operací databáze, distribuci k použití (např. zipfian) a velikost a počet polí v záznamu. Tyto vlastnosti jsou statické a mohou sloužit k výkonnostním testům pro různé databáze.

- Runtime vlastnosti – Vlastnosti specifické pro daný projekt. Například použití různých vrstev databázového rozhranní(např. pro MongoDB,CouchDB, HBase, atd.), vlastností k inicializaci těchto vrstev, počet klientských vláken, atd. Tyto vlastnosti jsou naopak dynamické a budou jiné pro každý projekt (např. databáze, cílová propustnost a další).



Obrázek č.13 Architektura YCSB klienta

6.1.1 Popis balíčků Yahoo! Cloud Serving Benchmark

V balíku src jsou uloženy další čtyři balíčky nezbytné pro samotný běh YCSB benchmarku.

- Základní je balíček com.yahoo.ycsb. Tento balíček obsahuje třídy jako com.yahoo.ycsb.DB která vykonává jednotlivé operace (read, update, atd.) v databázi a dále například třídu com.yahoo.ycsb.DBWrapper, která měří latenci a počítá návratové kódy a další metody. Jejich výčet je dlouhý všechny jsou uloženy v balíčku com.yahoo.ycsb s patřičnými komentáři.

- Další balíčkem je com.yahoo.ycsb.generator, která obsahuje třídy jako com.yahoo.ycsb.CounterGenerator, com.yahoo.ycsb.IntegerGenerator a další. V balíčku com.yahoo.ycsb.generator jsou všechny uloženy i s komentáři.

- Balíček com.yahoo.ycsb.measurements obsahuje třídy com.yahoo.ycsb.measurements.Measurements a další. V tomto balíčku třídy shromažďují a zaznamenávají veškeré naměřené hodnoty a histogramy a při požadavku je zasílají na výstup.

- Balíčekcom.yahoo.ycsb.workloads obsahuje pouze třídu com.yahoo.ycsb.workloads.CoreWorkload. V této třídě jsou definovány sady klientů vykonávající operace vytváření, čtení, aktualizace, a mazání.

6.2 Jádro souborů zatížení

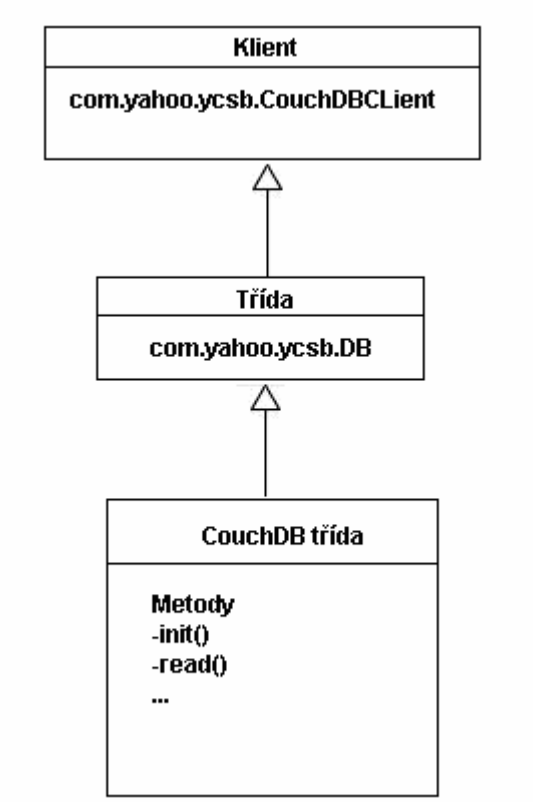
YCSB obsahuje sadu zatěžovacích souborů, které definují výkonnostní testy pro "cloud" systémy, lze také samozřejmě definovat vlastní zatěžovací soubory. Nicméně, jádro zatěžovacích souborů obsahuje dostatečně rozmanité zatěžovací soubory a výsledky z nich získané stačí k ohodnocení a porovnání různých databázových systémů.

Jádro souborů zatížení se skládá ze šesti různých souborů:

- Workload A: Update heavy workload – Tento soubor zatížení obsahuje 50 % čtecích operací a 50% zapisovacích operací. Dle komentářů v samotných souborech zatížení příkladem použití je relační úložiště zaznamenávající poslední akce.
- Workload B: Read mostly workload – Tento soubor zatížení obsahuje 95% čtecích operací a 5% zapisovacích operací. Dle komentářů v samotných souborech zatížení příkladem použití je označování fotografií, přidání označení je aktualizace, ale většina operací je určena pro čtení těchto označení.
- Workload C: Read only – Tenhle soubor zatížení je určen jen ke čtení. Dle komentářů v samotných souborech zatížení příkladem použití je systém Hadoop.
- Workload D: Read latest workload – V tomhle souboru zatížení, jsou nové záznamy vkládány a tyto nedávno a nejčastěji vkládané záznamy se stávají nejdůležitějšími. Dle komentářů v samotných souborech zatížení příkladem použití jsou aktualizace uživatelských stavů, kdy uživatelé chtějí číst ty nejnovější aktualizace.
- Workload E: Short ranges – V tomhle souboru, jsou dotazovány krátké intervaly záznamů, místo dotazování jednotlivých záznamů. Dle komentářů v samotných souborech zatížení příkladem použití je výměna informací mezi vlákny.
- Workload F: Read-modify-write – V tomto souboru zatížení, si klient přečte záznam, modifikuje ho a změny zpátky zapíše. Dle komentářů v samotných souborech zatížení příkladem použití je uživatelská databáze, kde jsou uživatelské záznamy čteny a modifikovány uživatelem, nebo k zaznamenávání činnosti uživatele.

6.3 Implementace vrstvy databázového rozhranní pro databázi CouchDB

Pro databázi CouchDB bylo nutné tuto vrstvu databázového rozhranní vytvořit. Základní třídou pro vrstvy databázového rozhranní pro všechny databáze je třída v balíčku com.yahoo.ycsb.DB. Je to abstraktní třída, tedy bylo nutné vytvořit novou třídu, která rozšiřuje třídu DB v balíčku com.yahoo.ycsb.DB. Tato třída má veřejný konstruktor bez argumentů.
Dále bylo nutné vytvořit metodu init(). Tato metoda je volána jednou za databázovou instanci, takže pokud je zde více vláken, každá databázová instance musí volat metodu init() samostatně. Tato metoda je využita k nastavení připojení k databázi a k provádění všech inicializačních akcí.
Dalším krokem bylo implementování samotných dotazovacích a aktualizačních metod (Read, Update, Insert, Delete, Scan). Tyto metody přebírají název tabulky a klíč daného záznamu. Pro metody čtení (Read a Scan) si metody navíc přebírají set polí, které mají být přečteny a poskytují strukturu (HashMap nebo vektor HashMap) k ukládání vrácených dat. Pro metody zapisování (Insert, Update) si metody přebírájí HashMapu, která přiřazuje názvy jednotlivých polí k hodnotám.



Obrázek č. 14 Architektura vrstvy databázového rozhranní

6.4 Samotné spuštění výkonnostního testu

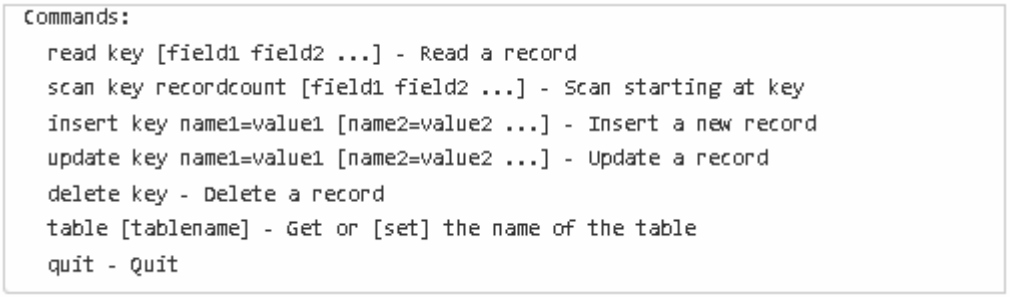
Samotné spuštění výkonnostního testu je rozděleno na šest nezbytných kroků. Níže je uvedeno podrobné rozebrání každého z těchto kroků.

6.4.1 Nastavení databázového systému k testování

Prvním krokem je nastavení databázového systému, který chceme testovat. To lze provést na jednom či více počítačích, v závislosti na konfiguraci kterou chceme testovat. Také musíme vytvořit nebo nastavit tabulky pro ukládání záznamů. Detaily nastavení se liší podle jednotlivých databázových systémů a závisí na souboru zatížení, který chceme spustit. Před spuštěním samotného YCSB klientu, musí být vytvořeny tabulky, protože klient sám o sobě vytvoření tabulky nevyžaduje.
Tabulky musí být vytvořeny v závislosti na souboru zatížení. Pro jádro souboru zatížení YCSB klient předpokládá, že existuje tabulka s názvem "usertable" s flexibilním schématem. Vrstva rozhranní databáze obdrží žádosti o čtení či zápis záznamů v tabulce a přeloží je na požadavek pro aktuální úložiště, které jsem alokovali. To může znamenat, že budeme muset poskytnout vrstvě rozhranní databáze informace k pochopení, které úložiště máme na mysli. Proto je nezbytné vytvořit rodinu sloupců (column family) a tuto rodinu nějak pojmenovat (např. values).

6.4.2 Výběr vrstvy databázového rozhranní

Vrstva databázového rozhranní je třída napsaná v jazyce Java, která vykonává operace čtení, vkládání, mazání a skenování generované YCSB klientem. Tato třída je podtřídou abstraktní databázové třídy v balíčku "com.yahoo.ycsb". Pokud zadáme jméno třídy v příkazovém řádku při spuštění YCSB klienta, tak klient dynamicky načte požadovanou třídu rozhranní. Všechny vlastnosti specifikované v příkazové řádce nebo v souborech parametrů jsou předány instanci databázového rozhranní a mohou být využity ke konfiguraci databázové vrstvy (například ji mohou sdělit hostname databáze, kterou chceme testovat).
Příkazy je možno spouštět přímo v databázi pomocí příkazu –ycsb. Tento klient využívá vrstvu databázového rozhranní k posílání příkazů do databáze. Klient také poskytuje jednotné rozhranní pro různé databáze a může být také použit ke kontrole dat v databázi).



Obrázek č.15 Specifikace vlastností v příkazovém řádku

6.4.3 Výběr souboru zatížení

Soubor zatížení definuje data, které mají být načteny do databáze během načítací fáze a operace, které budou provedeny se souborem dat během transakční fáze. Soubor zatížení je kombinací třídy Workload Java (podtřídou balíčku com.yahoo.ycsb.Workload) a souboru parametrů (ve formátu Java Properties).
Protože vlastnosti datového souboru musí být známy během načítací fáze (tak aby mohl být vytvořen a vložen správný typ záznamu) a během transakční fáze (tak aby mohlo být uvedeno správné ID záznamu a pole), tak jeden soubor vlastností je sdílen mezi oběma fázemi, tedy i soubor parametru je použit oběmi fázemi. Workload Java třída používá tyto vlastnosti buď na vkládání záznamů (načítací fáze) nebo k vykonání transakcí proti těmto záznamům (transakční fáze). Volba, kterou fázi spustit je založena na parametru, který specifikujeme při spuštění příkazu –ycsb.
Při spuštění YCSB klienta můžeme v příkazovém řádku specifikovat jak třídu Java tak soubor parametrů. Klient dynamicky načte třídu souboru zatížení, předá jí vlastnosti ze souboru parametru (a všechny další vlastnosti specifikované v příkazovém řádku) a poté začne vykonávat samotný soubor zatížení.

6.4.4 Výběr Runtime parametrů

Ačkoli třída souboru zatížení a soubor parametru určují konkrétní soubor zatížení, jsou zde dodatečná nastavení, které lze specifikovat pro konkrétní běh výkonnostního testu. Tato nastavení můžeme zadat do příkazové řádky při spuštění YCSB klienta.

Tyto nastavení jsou:

- -threads: Počet klientských vláken. Implicitně YCSB klient využívá jedno pracovní vlákno
- -target: Cílový počet operací za vteřinu. Implicitně YCSB klient vykonává tolik operací kolik může.
- -s: Status pro dlouho běžící soubory zatížení. Klient bude vypisovat status každých 10 sekund

6.4.5 Načtení dat

Jak už bylo řečeno soubory zatížení mají dvě fáze, načítací a transakční. Chceme li načíst data, spustíme YCSB klienta a sdělíme mu, že má začít tuto fázi provádět.

```
$ ./bin/ycsb load basic -P workloads/workloada
```

Obrázek č.16 Načtení souboru zatížení (Workload A)

Vysvětlení příkazů:

- load – tento parametr sdělí klientovi, že má vykonat načítací fázi souboru zatížení
- basic – parametr basic sdělí klientovi, že má použít základní databázovou vrstvu.
- -P -parametr –P je použit k načtení souboru vlastností. V tomto případě je načten soubor vlastností uložených v souboru zatížení (workload A).

Obecně, je lepší ukládat všechny důležité vlastnosti do nového souboru parametrů, namísto jejich zadávání v příkazové řádce.

6.4.6 Spuštění souboru zatížení

Jakmile jsou data načteny, můžeme spustit samotný soubor zatížení. To provedeme tím, že klientovi sdělíme, že má spustit transakční fázi daného souboru zatížení. Hlavním rozdílem je použití parametru *run* namísto parametru *load.*
Výše bylo zmíněno, že pro kontrolu množství načítaných dat používáme parametry *–threads* a *–target*. Pokud máme 10 vláken zvládajících celkem 100 operací za vteřinu (10 op/s na vlákno) a průměrná latence operací není vyšší než 100 ms bude každé vlákno schopno provádět požadovaných 10 operací za sekundu. Obecně potřebujeme tolik vláken, aby každé z nich bylo schopno zvládnout tolik operací kolik je požadováno, jinak bude dosažená propustnost menší než specifikovaná cílová propustnost.

```
$ ./bin/ycsb run basic -P workloads/workloada -P large.dat -s -threads 10 -target 100 > transactions.dat
```

Obrázek č.17 Spuštění souboru zatížení s parametry

Výše na obrázku jsou použity parametry v příkazovém řádku *–threads 10* a *–target 100*, tedy že je použito 10 vláken a že je specifikováno 100 op/s na cíl.
Pokud proběhne vše v pořádku, tak na konci klient zašle statistiky o výkonu na standardní výstup. V předchozím příkladu budou tyto statistiky zapsány do souboru transaction.dat. Implicitně se zaznamenává propustnost, průměrná, minimální, maximální, 95th a 99th percentil latence, počet návratových kódů a histogram latencí pro každý typ operace (čtení, zápis, atd.). Návratové kódy jsou definovány vrstvou databázového rozhranní a slouží ke zjištění zda nedošlo k nějaké chybě.

```
[OVERALL],RunTime(ms), 10110
[OVERALL],Throughput(ops/sec), 98.91196834817013
[UPDATE], Operations, 491
[UPDATE], AverageLatency(ms), 0.054989816700611
[UPDATE], MinLatency(ms), 0
[UPDATE], MaxLatency(ms), 1
[UPDATE], 95thPercentileLatency(ms), 1
[UPDATE], 99thPercentileLatency(ms), 1
[UPDATE], Return=1, 491
[UPDATE], 0, 464
[UPDATE], 1, 27
[UPDATE], 2, 0
[UPDATE], 3, 0
[UPDATE], 4, 0
...
```

Obrázek č. 18 Příklad výstupu

Tento výstup ukazuje: Celkový čas 10.11 sekund, průměrná propustnost 98.9 op/s, 491 operací aktualizace s průměrnou, min, max, 95th a 99th percentilem latencí. 491 aktualizací mělo návratový kód jedna, 464 operací bylo dokončeno za méně než 1ms a 27 bylo dokončeno mezi 1 a 2 ms.

6.5 Shrnutí

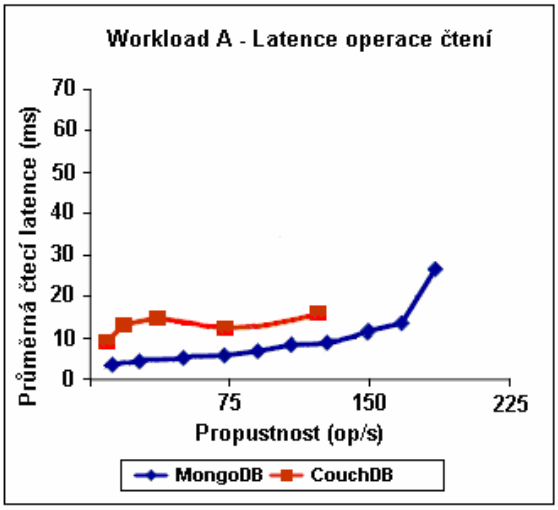
Jedním z důvodů proč jsem si zvolil tento benchmark je kvůli velké popularitě, které se těší nejen u vývojářů databází ale v celé komunitě těch, kteří potřebují nějakým způsobem otestovat svůj databázový systém. Pomocí tohoto výkonnostního testu lze otestovat všechny důležité vlastnosti databází jako např. latenci a propustnost při operacích čtení, vkládání, atd. Druhým a podstatnějším důvodem je, že vrstva databázového rozhranní nepotřebuje znát detaily o konkrétní databázi.To umožňuje klientovi generovat operace jako čtení a aktualizace záznamů beztoho aniž by musela rozumět konkrétnímu API databáze. Proto je relativně snadné testovat nové databázové systémy, stačí vytvořit vrstvu databázového rozhranní pro konkrétní databázi v mém případě pro CouchDB.

7. Výsledky výkonnostního testu

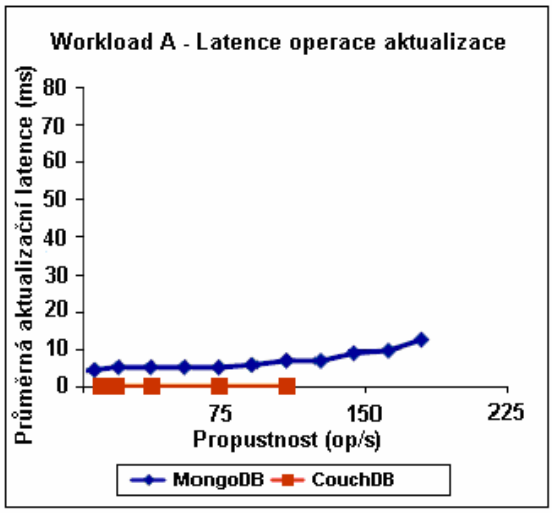
Samotné testování probíhalo při použití pěti vláken a při 100000 záznamech a bylo použito pět uzlů, každý uzel měl počítačovou sestavu Intel Xeon CPU X5670 2.93 GHz, 1Gb RAM Vrstva databázového rozhranní pro MongoDB je obsažena v samotném YCSB benchmarku. Pro databázi CouchDB bylo nutné tuto vrstvu databázového rozhranní vytvořit a přesunout do balíčku com.yahoo.ycsb.

7.7.1 Výsledky Workload A

Nejprve jsem testoval obě databáze zatěžovacím souborem A, který obsahuje 50 procent čtecích operací a 50 procent zapisovacích operací. Na obrázku č. 19 jsou znázorněny křivky latence v závislosti na propustnosti pro oba databázové systémy a pro obě operace čtení a zápisu. U obou případů jsem zvyšoval propustnost do té doby než se její růst zastavil. Jak obrázek ukazuje latence operací se zvyšovala společně se zvyšující se propustností. Obrázek č. 19 a) ukazuje, že MongoDB dosáhla čtecí propustnosti 179,49 op/s a latence 28ms a CouchDB dosáhla čtecí propustnosti pouze 134,089 op/s s latencí 17ms. Obrázek č. 19b) ukazuje, že MongoDB dosáhla aktualizační propustností 176,28 op/s a latencí 13 ms. CouchDB dosáhla aktualizační propustnosti 112,764 op/s ovšem CouchDB dosahuje překvapivě nízké latence při všech aktualizačních operacích průměrně 3 ms. Je to způsobeno tím, že CouchDB používá obousměrnou replikaci, tudíž každý uzel může zapisovat data do databáze. MongoDB používá Master-Slave replikaci, tedy číst mohou všechny uzly ale zapisovat smí pouze řídící. Díky tomu databáze MongoDB má vyšší latenci pro zapisovací a aktualizační operace



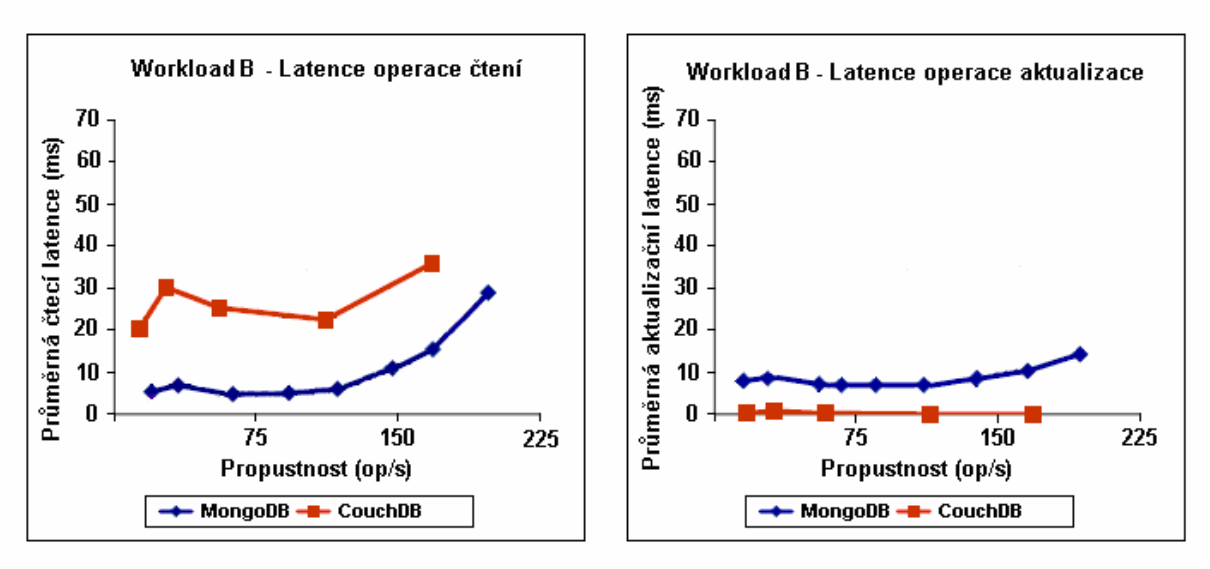
Obrázek č. 19 a) Výsledky workload A



Obrázek č. 19 b) Výsledek workload A

7.7.2 Výsledky Workload B

Zatěžovací soubor B obsahuje 95 procent čtecích operací a 5 procent zapisovacích operací. Výsledky jsou znázorněny na obrázku č. 20. CouchDB nebyla schopna dosáhnout propustnosti při čtení 200 op/s. Navíc čtecí latence se prudce zvyšovala (37ms), když se propustnost zvýšila ze 111,26 na 157,012 op/s jak ukazuje obrázek č. 20 a). Aktualizační propustnost dosáhla hodnoty 162,380 op/s při latenci 3 ms. MongoDB dosáhla propustnosti 191,865 op/s s čtecí latencí 31ms a aktualizační latencí 16ms.

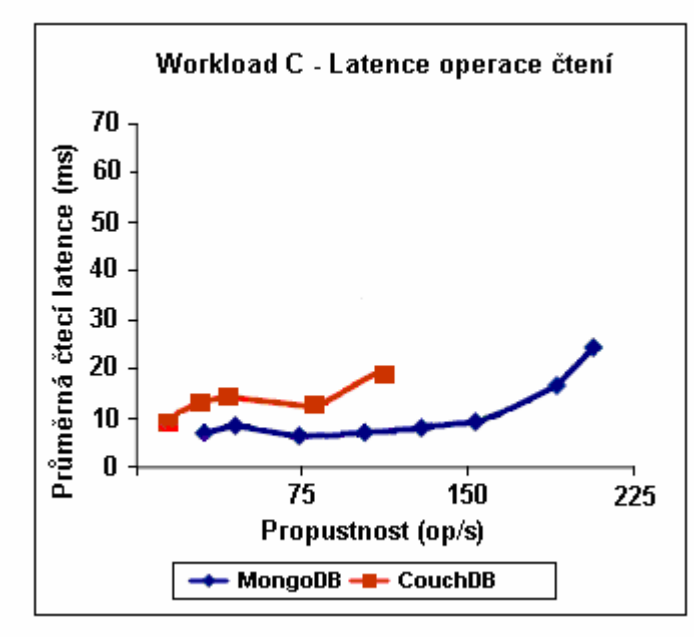


Obrázek č. 20 a) Výsledky workload B

Obrázek č. 20 b) Výsledek workload B

7.7.3 Výsledky Workload C

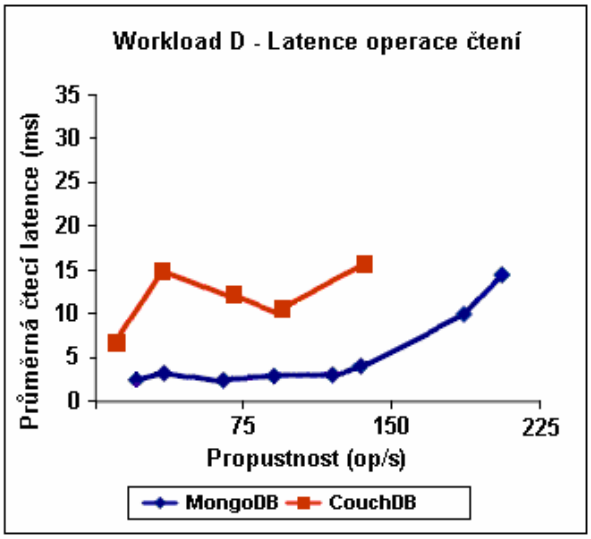
Zatěžovací soubor C je určen jen ke čtení. Na obrázku č. 21 vidíme, že MongoDB je schopna dosáhnout nejvyšší propustnosti (202,063 op/s) a průměrnou čtecí latencí 25,89 ms. CouchDB byla schopna dosáhnout nejvyšší propustnosti (113,24 op/s) s průměrnou čtecí latencí 20,22 ms.
Je třeba poznamenat, že MongoDB načte 8KB záznamů do paměti s každým čtecím požadavkem, zatímco CouchDB načte 32KB záznamů s každým čtecím požadavkem. Vzhledem k tomu, že vstupní a výstupní činnost v tomto souboru zatížení používá z velké části náhodný přístup, CouchDB mrhá datovou propustností čtením dat, které nejsou potřeba.



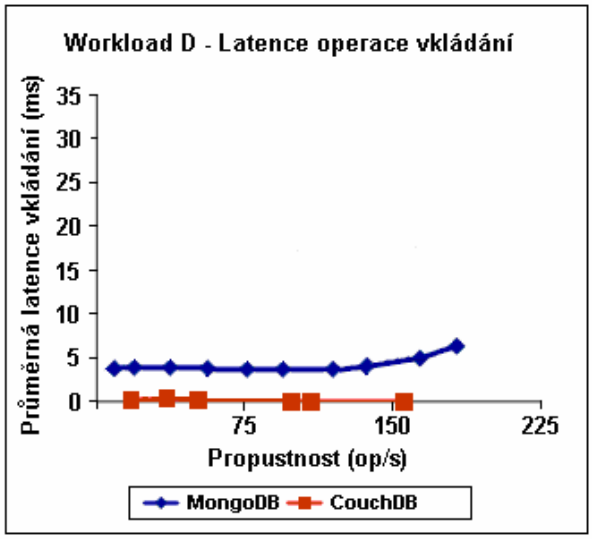
Obrázek č. 21 Výsledky workload C

7.7.4 Výsledky Workload D

Zatěžovací soubor D je velmi podobný zatěžovacímu souboru B. Obsahuje 95 procent čtecích operací a 5 procent operací vkládání. Obrázek č.22 ukazuje latenci operací vkládání a čtení. V tomto souboru mají všechny čtecí požadavky charakter "Číst nejnovější". Díky tomu je zde velká pravděpodobnost, že požadavek na čtení přečte nejnovější záznam, který byl právě vložen do databáze. Dále při provádění tohoto zátěžového souboru MongoDB vykazuje vyšší latenci pro nízké hodnoty propustnosti (do 37,498 op/s) ve srovnání s vyššími hodnotami propustnosti (68,63 op/s a 120,634 op/s). Toto chování je způsobeno tím, že pro hodnoty s nižší propustností není paměť zcela naplněna požadovanými daty. Pro příklad při propustnosti 35 op/s je paměť naplněna daty jen z poloviny a díky tomu hodně čtecích požadavků bude vyžadovat dodatečné I/O operace disku.



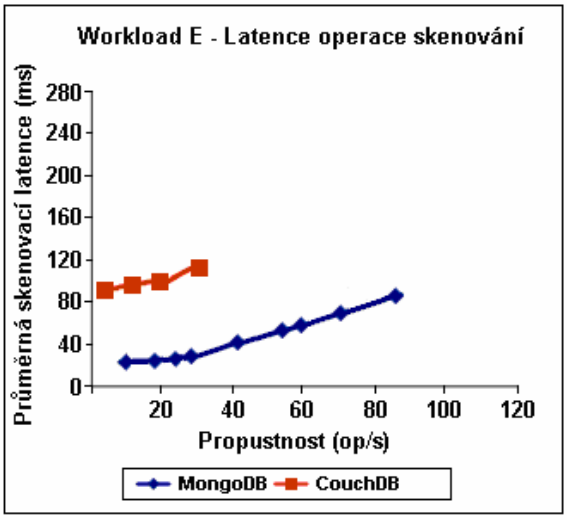
Obrázek č. 22 a) Výsledky workload D



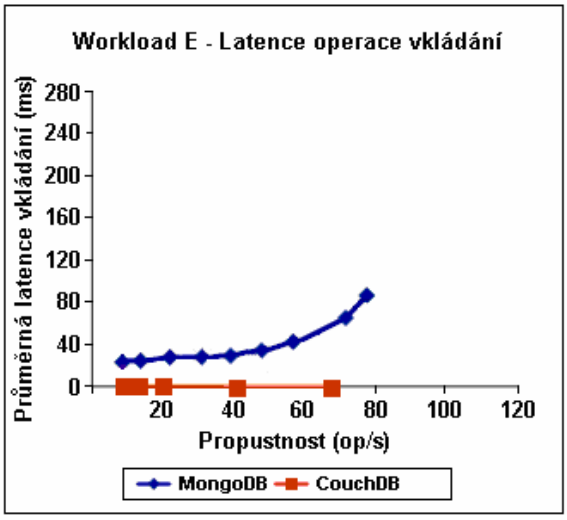
Obrázek č. 22 b) Výsledek workload D

7.7.5 Výsledky Workload E

Obrázek č. 23 ukazuje výkon obou databázových systémů v zatěžovacím souboru E, kde jsou dotazovány krátké intervaly záznamů. Jak obrázek č. 23a) ukazuje, MongoDB dosahuje nejvyšší propustnosti 88,164 op/s při skenovací latenci 88ms. MongoDB používá rozdělování dat po intervalech k distribuci datových bloků mezi servery. To znamená, že MongoDB může stanovit, na základě intervalové žádosti, které oddíly obsahují požadovaná data a skenovat pouze je (obvykle jeden oddíl pro dotazování každého krátkého intervalu) zatímco CouchDB musí skenovat tolik oddílů kolik potřebuje, dokud nejsou požadované záznamy nalezeny.



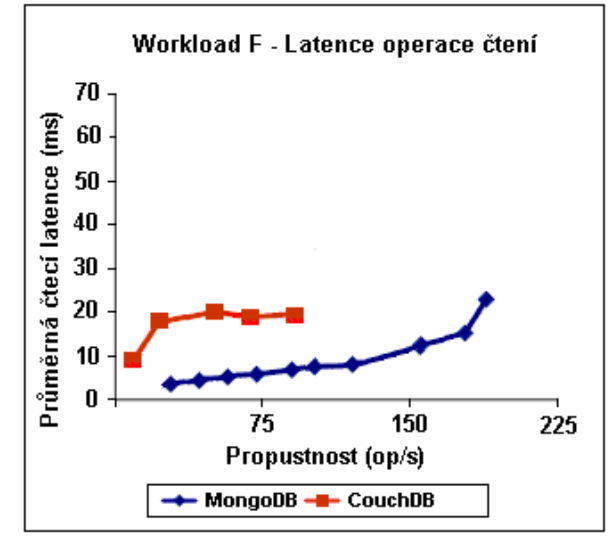
Obrázek č. 23 a) Výsledky workload E



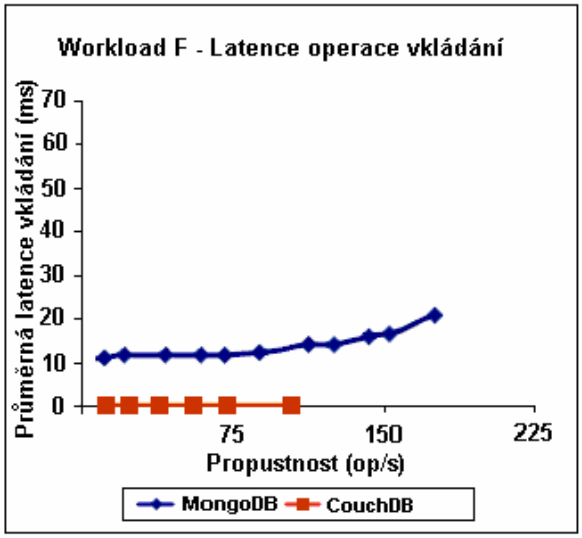
Obrázek č. 23 b) Výsledek workload E

7.7.6 Výsledky Workload F

Zatěžovací soubor F simuluje často používaný model čtení záznamu, jeho modifikaci a následné zapsání změn zpět do databáze. Tento zatěžovací soubor je velmi podobný zatěžovacímu souboru A (50 procent čtecích operací a 50 procent zapisovacích operací) s výjimkou toho, že aktualizace mají charakter "číst-modifikovat-zapsat" nejsou to tedy jen jednoduché operace zapisování. Jako ve všech předchozích případech MongoDB dosáhla vetší propustnosti 177,683 op/s při čtecí latenci 24 ms obrázek č.24a) a propustnosti 172,29 op/s při vkládací latenci 21 ms obrázek č.24b). CouchDB dosáhla největší propustnosti 92,368 op/s při čtecí latenci 22 ms a propustnosti 106,56 op/s při latenci vkládání 2ms.



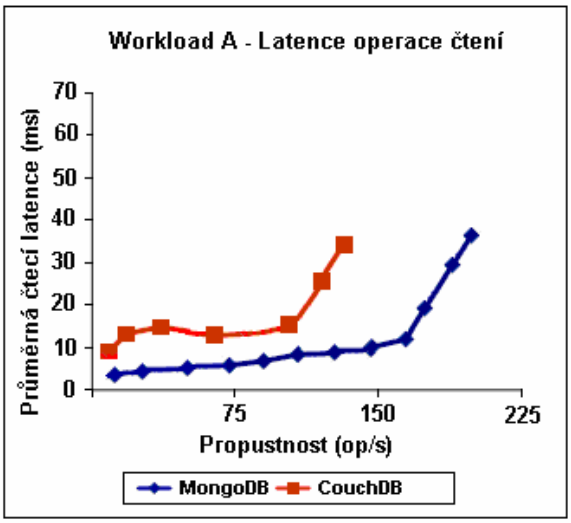
Obrázek č. 24 a) Výsledky workload F



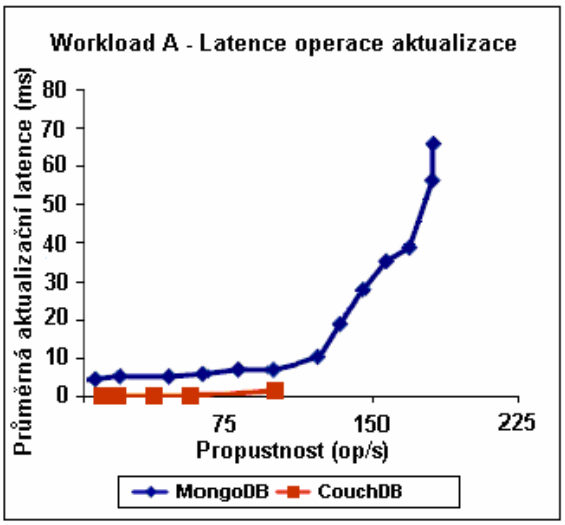
Obrázek č. 24 b) Výsledek workload F

7.7.7 Výsledek Workload A Experiment

Tento experiment spočíval v tom, že jsem testoval obě databáze zatěžovacím souborem A. Po uplynutí jedné hodiny (celý test trval 1 hodinu a 32 minut) jsem odstavil dva z pěti severů. Jak můžete sami vidět na obrázku č. 25a po odstavení serverů u obou databází došlo ke zvýšení čtecí latence. Obrázek č. 25b ukazuje, že u databáze MongoDB po odstavení obou serverů, došlo k rapidnímu zvýšení aktualizační latence až na 68ms, je to díky použití Master – Slave replikace kdy zapisovat smí pouze řídící uzel. Naopak CouchDB používá obousměrnou Master – Master replikaci, tedy zapisovat mohou všechny uzly a shození dvou serverů nepředstavuje z hlediska aktualizační latence až takový problém.



Obrázek č. 25 a) Výsledky workload A



Obrázek č. 25 b) Výsledek workload A

Z grafů je patrno, že databáze MongoDB dosáhlo při použití všech souborů zatížení lepších výsledků než databáze CouchDB. MongoDB má mírný náskok v Get / Put souborech zatížení (tedy Workloads A-D a F) ovšem je značně rychlejší pro vyhledávací soubor zatížení (tedy Workload E). MongoDB používá pro ukládání dat soubory přímo mapované do paměti, umožňuje rozčlenit data do kolekcí, používá datový formát BSON pro urychlení výpočtů a dále MongoDB používá na rozdíl od CouchDB map/reduce jen pro zpracování dat ale ne pro tradiční dotazování.

8. Srovnání MongoDB a CouchDB

Je celkem obtížné odpovědět na otázku v čem se vlastně MongoDB a CouchDB liší. Obě databáze jsou dokumentově orientované, nemají žádné schéma, používají stejný datový formát pro reprezentaci dat (JSON) a obě používá stále se zvětšující počet vývojářů. Přese všechno se opravdu v některých ohledech liší, níže se pokusím vysvětlit v čem.

8.1 Dotazování

Jednou z věcí v čem je databáze MongoDB lepší je dotazování. CouchDB používá chytrý systém pro generování indexů, které podporují jednotlivé dotazy. Je to elegantní přístup ale jeden index musí předeklarovat tyto struktury pro každý dotaz, který má být vykonán.
MongoDB používá dynamické dotazování pomocí kterého může vytvářet dotazy bez použití indexů. MongoDB obsahuje nástroj na optimalizování dotazů, který se o to stará. Je pohodlné mít možnost administrativní kontroly dat a při tom nepoužívat indexy. Pokud ovšem index dokonale odpovídá dotazu, přístupy CouchDB a MongoDB jsou si pak koncepčně podobné.

8.2 Úložiště

CouchDB vlastně neukládá svá data do formátu JSON ale ukládá se jako formát jazyka Erlang (obvyklý binární formát) a při každé čtecí/zapisovací operaci se teprve převádí na formát JSON. , Zatímco MongoDB převádí formát JSON na BSON(Binary JSON). Hlavní výhodou BSON oproti JSON je výkonnost (výpočty probíhají daleko rychleji a data mají menší velikost). Právě kvůli rychlosti klientské ovladače převádí data přímo do formátu BSON.

8.3 Kolekce

Databáze CouchDB je jedno obrovské datové úložiště, zatímco databáze MongoDB umožňuje rozčlenit data do kolekcí podobných SQL tabulkám. Pokud používáme kolekce k logickému rozdělení dat, tak rychlost dotazování se podstatně zvýší pokud budeme provádět dotazy na podmnožině dat než nad všemi daty.

8.4 Replikace

Replikace v databázích CouchDB a MongoDB se velmi liší. CouchDB je určena jen pro určitý typ aplikací, protože podporuje rychlé Master - Master replikace a synchronizaci mezi vzdálenými CouchDB databázemi. Každá databáze v CouchDB je master. CouchDB je dobrá volba pokud je velká část aplikace synchronní mezi databázemi, které mohou kdykoli přejít do režimu offline nebo online a vyžadují řešení konfliktů a souběhů.
Naproti tomu MongoDB nepoužívá Master - Master replikaci, replikace je jen určena pro redundanci ve více konvenčních centralizovaných databázích. MongoDB používá topologii zvanou

Replica Sets. Tento přístup je velmi podobný klasické master-slave replikaci, ale s automatickou obnovou po pádu systém. Jestliže dojde k výpadku primárního uzlu cluster automaticky vybere jeden ze sekundárních uzlů a povýší ho na uzel primární. Dále CouchDB používá obousměrnou replikaci, tudíž každý uzel může zapisovat data do databáze naproti tomu MongoDB používá Master-Slave replikaci, tedy číst mohou všechny uzly ale zapisovat smí pouze řídící.

8.5 Přístup

CouchDB používá HTTP na modelu REST založené rozhranní. Toto rozhranní je intuitivní a velmi dobře navržené. Také umožňuje psaní JavaScriptově založené aplikace, které se volají přímo do databáze z klientského prohlížeče.
Naproti tomu MongoDB používé pouze binární protokol.

8.6 JavaScript

Obě databáze CouchDD i MongoDB využívají JavaScript. Couchdb používá JavaScript extenzivně a dokonce i při budování pohledů.
MongoDB také podporuje používaní JavaScriptu ale spíše jako doplněk. Dotazy v MongoDB jsou typicky vyjádřeny jako dotazovací objekty ve formátu JSON ale lze je i vyjádřit pomocí JavaScript výrazu, který bude součástí dotazu. MongoDB také podporuje spuštění doplňkových JavaScriptových funkcí na straně serveru a také pro map/reduce operace.

8.7 Shrnutí

I když podle výkonnostních testů se může zdát, že MongoDB je ve všech směrech lepší volbou, není tomu tak.
CouchDB a MongoDB jsou optimalizovány pro odlišné případy použití. Pokud potřebujeme podporu dynamického dotazování nebo je pro nás důležitý faktor rychlosti, tak pro vývoj konvenčních webových aplikací, které vyžadující redundantní datové úložiště s automatickou obnovou po pádu systému je lepší volbou použití databáze MongoDB.
CouchDB je mnohem více než jen open-source projekt. Z hlediska replikace je databáze CouchDB vhodná pro vývoj centralizovaných aplikací pro správu údajů, které mají webově založené rozhranní a mobilního klienta, který dovoluje offline prohlížení a pro data, které potřebují synchronizaci mezi dvěma databázemi.
Zkrátka pokud potřebujeme master-master replikaci nebo požadujeme maximální trvanlivost jediného serveru z důvodu jen jediného databázového serveru pak je vhodný kandidát CouchDB.
Pokud požadujeme maximální čtecí/zapisovací propustnost nebo potřebujeme ukládat obrovské datové soubory pak je volbou MongoDB.

9. Závěr

Cílem této bakalářské práce bylo otestování dvou databázových systémů CouchDB a MongoDB prostřednictvím testovacího nástroje Yahoo! Cloud Serving Benchmark a následného porovnání jejich vlastností.
Ve své práci jsem nejprve specifikoval základní rysy relačního databázového modelu a relačních databází a jejich problémů.
Poté je část věnovaná NoSQL databázím. Je zde nastíněna historie NoSQL databází a vysvětlen samotný pojem NoSQL. V této kapitole jsou též popsány kategorie NoSQL a jejich základní rysy.
Další část je věnovaná samotné databázi MongoDB, kde jsem se snažil popsat tuto databázi, její vlastnosti a způsob jakým pracuje s daty. Jsou zde uvedeny příklady datového modelu, způsob dotazování, replikace a další.
Následuje část věnovaná databázi CouchDB, je zde uvedena její historie a také popis vlastností a nástrojů jež jsou její součástí.
V další části je představen testovací nástroj Yahoo! Cloud Serving Benchmark. Je zde popsána jeho architektura, funkce a parametry potřebné ke spuštění výkonnostního testu. Také jsou zde popsány jednotlivé soubory zatížení.
Poslední část je věnována porovnání jednotlivých databázových systémů. Bylo použito několik kritérií srovnávání např. podle způsobu dotazování, replikace, úložiště atd.
Také jsou zde uvedeny samotné výsledky výkonnostního testu jejich datové propustnosti. Poté je zde uvedeno v čem jednotlivé databáze vynikají a k jakým případů použití jsou určeny.

8.1 Přínos

Hlavním přínosem je to vytvoření samotné vrstvy databázového rozhranní pro databázi CouchDB k jejímu otestování. Dále práce poskytuje přehled a srovnání jednotlivých databázových systémů včetně zdůraznění jejich hlavních myšlenek a rozdílů. Ze srovnání vyplývá, že ač každý z databázových systémů je určen pro trochu odlišné aplikační oblasti (CouchDB je vhodná pro vývoj centralizovaných aplikací pro správu údajů, které mají webově založené rozhranní a mobilního klienta, MongoDB pro vývoj konvenčních webových aplikací, které vyžadující redundantní datové úložiště s automatickou obnovou po pádu systému) tak oba tyto systémy mají mnoho společných vlastností. Záleží tedy pouze na tom pro jaké účely chceme tyto databáze využívat.

9. Literatura

[1] Rebecca M. Riordan, Vytváříme relační databázové aplikace (2000), 1. vydání, ISBN – 80-7226-360-9

[2] Overview of NoSQL Databases, http://newtech.about.com/od/databasemanagement, New Tech (2012)

[3] Kyle Banner, MongoDB in action (2011), ISBN - 9781935182870

[4] http://www.mongodb.org, 10Gen (2011)

[5] J. Chris Anderson, Jan Lebnardt (2010): O´reilly CouchDB the definitive guide, ISBN - 978-0-596-15589-6

[6] http://research.yahoo.com/ , Yahoo (2012)

[7] http://blog.msdn.com, (2011)

10. Přílohy bakalářské práce

1) YCSB benchmark s klienty pro MongoDB a CouchDB
2) Disk CD obsahující elektronické verze dokumentů bakalářské práce